(Director, Khateeb Institute of Technical Education)

Sunville Properties Database management Sys

Submitted by:

Gaurav Suvarna

Submitted to:

Sunville Properties

Under the Guidance of:

Prof. Junaid Khateeb (Director, Khateeb Institute of Technical Education)

Date of Submission: 10th July 2020.

Certificate Of completion

This is to certify that, Mr ________________has successfully implemented an application _______________.

The Application has been accepted as a completed project as it meets all the requirements specified.

12th July 2020

____

(Khateeb Institute of Technical Education)

Acknowledgements

I would like to express my sincere gratitude to thank my professors and supervisors & for providing their proper and invaluable guidance, comments and suggestions throughout the course of the project. I would specially thank for constantly motivating me to work harder. Also, I would like to thank for his assistance for the code & for his help during the preparation of the sample, for providing me an overview of the entire project.

Table of Contents

INTRODUCTION OF THE PROJECT

1. OBJECTIVE

Sunville Properties is a Colorado based property consultancy firm. They have appointed their agents across Major Cities around the world. They have sub-Companies which take care the business in different countries and are placed in the countries from where they operate from. The Company currently has been using multiple forms of data storage and want to streamline their working using an application, which can help them seamlessly navigate via different forms of storage. Also, the company seeks some insights into the current data and also going further in future. So, it has requested specific modules to be introduced in the system.

Section 1 :

System Requirement Specifications:
So, we had to build an application for a real estate company “Sunville Properties” which can help them keep a track of their business and help them grow accordingly. To make that happen, we have to provide them with a proper visual interface so that only their agents can access their data.

1) A Visual Interface to add the data inside each of their tables. A login authentication is mandatory for anyone to be able to modify the data.

2) The company needs an order look up (i.e. search) based on the following criteria,

a) Order number

b) Order Date

c) Customer code
kindly note: the company might use either one or all of them together at a time.
3) Generate a report that highlights the balance amounts for all orders in descending order. Do mention the name and code of the agent handling the order. This information needs to be updated in the database.

4) Which is the country with maximum number of registered customer and what is the collective payment amount and outstanding amount for all these customers collectively.

The company needs the following insights
1) On selection of the year, system should help them get the following

a) The total property area sold vs total property are leased in Sq-M only.

b) Of the years 2017,2018,2019- which year got maximum leased area in CA and WS countries.

c) What are the Agent codes of all the agents who have got deals in ‘OWNED’ categories across the years.
d) For the city of Chilliwack, which agent has got the maximum deals in leased form.

e) Compare the performance of all agents based on the area leased and owned for the years 2017,2018 and 2019. Who has been the best performer?

f) The Company seeks a time series analysis report of the orders received.

(in this section explain in details, what is the system to be designed, requirements and the desired results)

Section 2:

Technology used
We have made use of python programming language to build the application While building this project we have extensively used the softwares such as PyCharm (python interpreter), Xamp server (for handling database) and the internet.

From Python module we have used Tkinter to build the GUI for the application

Section 3:

Data Provided by the client:
We were given a dataset of a Real Estate Company called Sunville Properties We were also provided with all the properties sold in the month July from years 2017,2018,2019,2020.
We were also provided with the company details of the agents, orders, company, customers photos of which are as follow

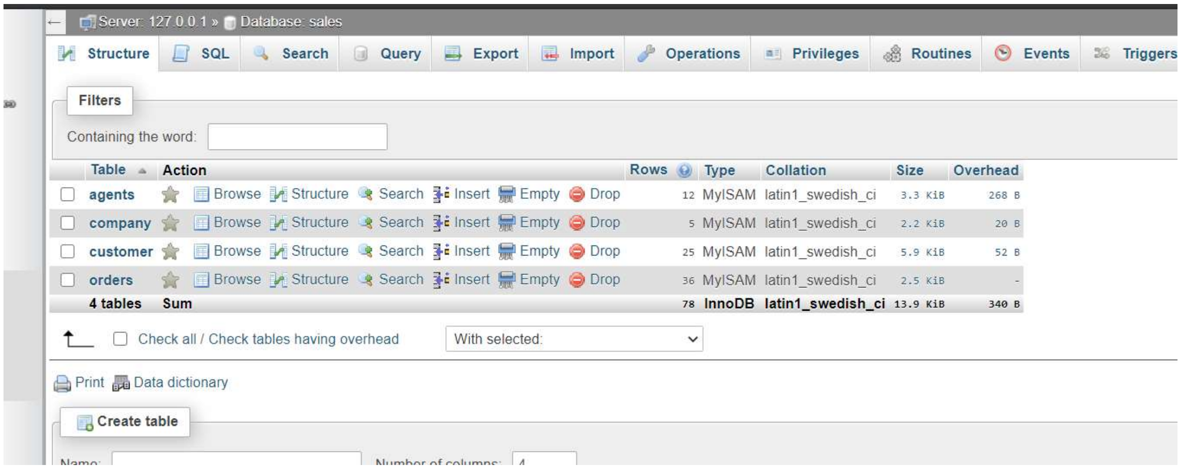
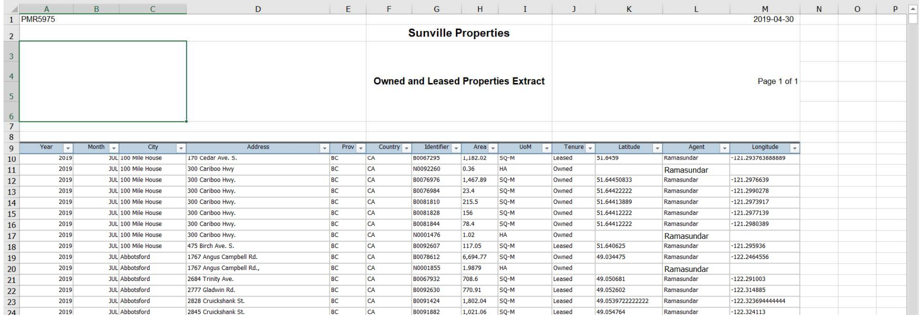


Table	Action	Rows	Type	Collation	Size	Overhead
agents	Browse Structure Search Insert Empty Drop	12	MyISAM	latin1_swedish_ci	3.3 KiB	268 B
company	Browse Structure Search Insert Empty Drop	5	MyISAM	latin1_swedish_ci	2.2 KiB	20 B
customer	Browse Structure Search Insert Empty Drop	25	MyISAM	latin1_swedish_ci	5.9 KiB	52 B
orders	Browse Structure Search Insert Empty Drop	36	MyISAM	latin1_swedish_ci	2.5 KiB	-
4 tables	Sum	78	InnoDB	latin1_swedish_ci	13.9 KiB	340 B



PMR5975 2019-04-30

Sunville Properties

Owned and Leased Properties Extract

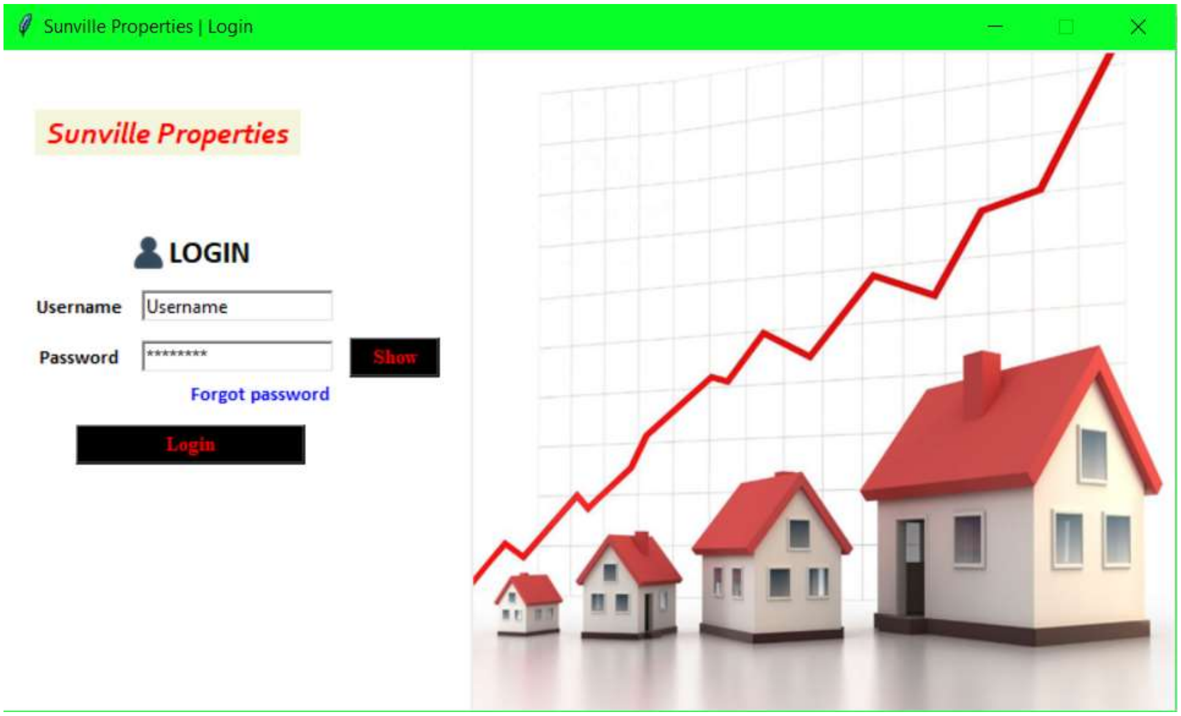
Page 1 of 1

Year	Month	City	Address	Prov	Country	Identifier	Area	UoM	Tenure	Latitude	Agent	Longitude
2019	JUL	100 Mile House	170 Cedar Ave. S.	BC	CA	B0067295	1,182.02	SQ-M	Leased	51.6459	Ramasundar	-121.293763888889
2019	JUL	100 Mile House	300 Cariboo Hwy	BC	CA	N0092260	0.36	HA	Owned		Ramasundar	
2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0076976	1,467.89	SQ-M	Owned	51.64450833	Ramasundar	-121.2976639
2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0076984	23.4	SQ-M	Owned	51.64422222	Ramasundar	-121.2990278
2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0081810	215.5	SQ-M	Owned	51.64413889	Ramasundar	-121.2973917
2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0081828	156	SQ-M	Owned	51.64412222	Ramasundar	-121.2977139
2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0081844	78.4	SQ-M	Owned	51.64412222	Ramasundar	-121.2980389
2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	N0001476	1.02	HA	Owned		Ramasundar	
2019	JUL	100 Mile House	475 Birch Ave. S.	BC	CA	B0092607	117.05	SQ-M	Leased	51.640625	Ramasundar	-121.295936
2019	JUL	Abbotsford	1767 Angus Campbell Rd.	BC	CA	B0078612	6,694.77	SQ-M	Owned	49.034475	Ramasundar	-122.2464556
2019	JUL	Abbotsford	1767 Angus Campbell Rd.,	BC	CA	N0001855	1.9879	HA	Owned		Ramasundar	
2019	JUL	Abbotsford	2684 Trinity Ave.	BC	CA	B0067932	708.6	SQ-M	Leased	49.050681	Ramasundar	-122.291003
2019	JUL	Abbotsford	2777 Gladwin Rd.	BC	CA	B0092630	770.91	SQ-M	Leased	49.052602	Ramasundar	-122.314885
2019	JUL	Abbotsford	2828 Cruickshank St.	BC	CA	B0091424	1,802.04	SQ-M	Leased	49.0539722222222	Ramasundar	-122.323694444444
2019	JUL	Abbotsford	2845 Cruickshank St.	BC	CA	B0091882	1,021.06	SQ-M	Leased	49.054764	Ramasundar	-122.324113

Section 4:

Screenshots:

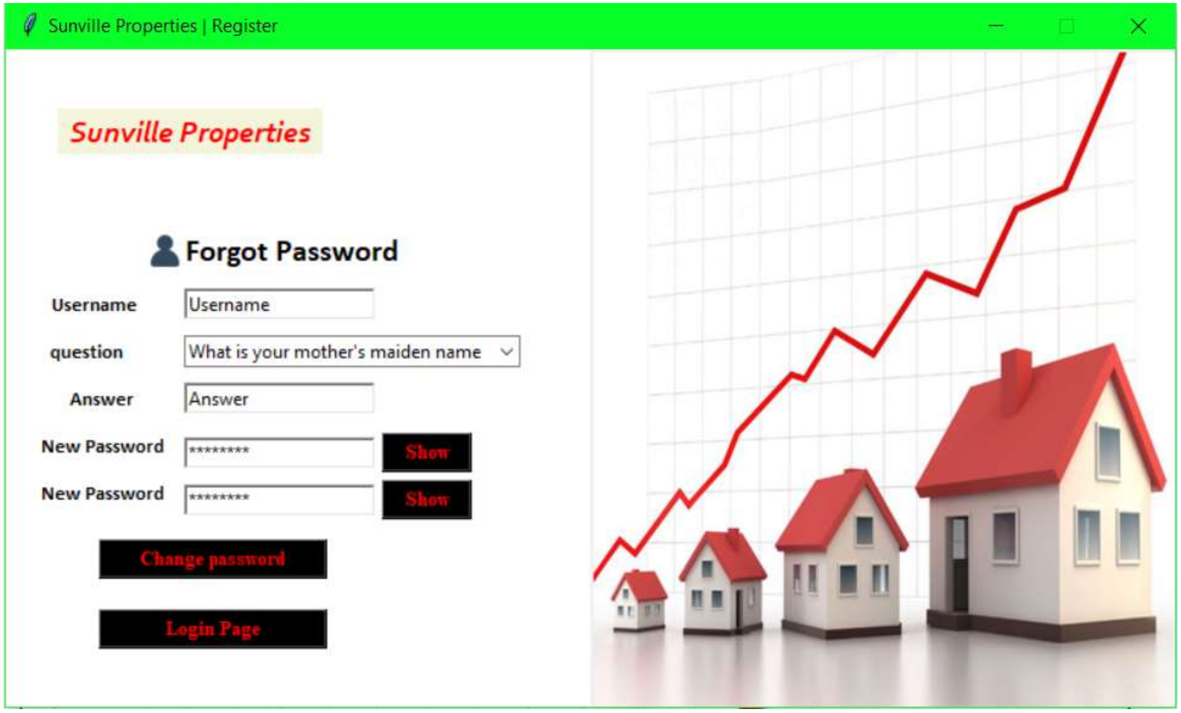
Login page:



Username: Username

Password: ********

Forgot Password Page:



Username: Username

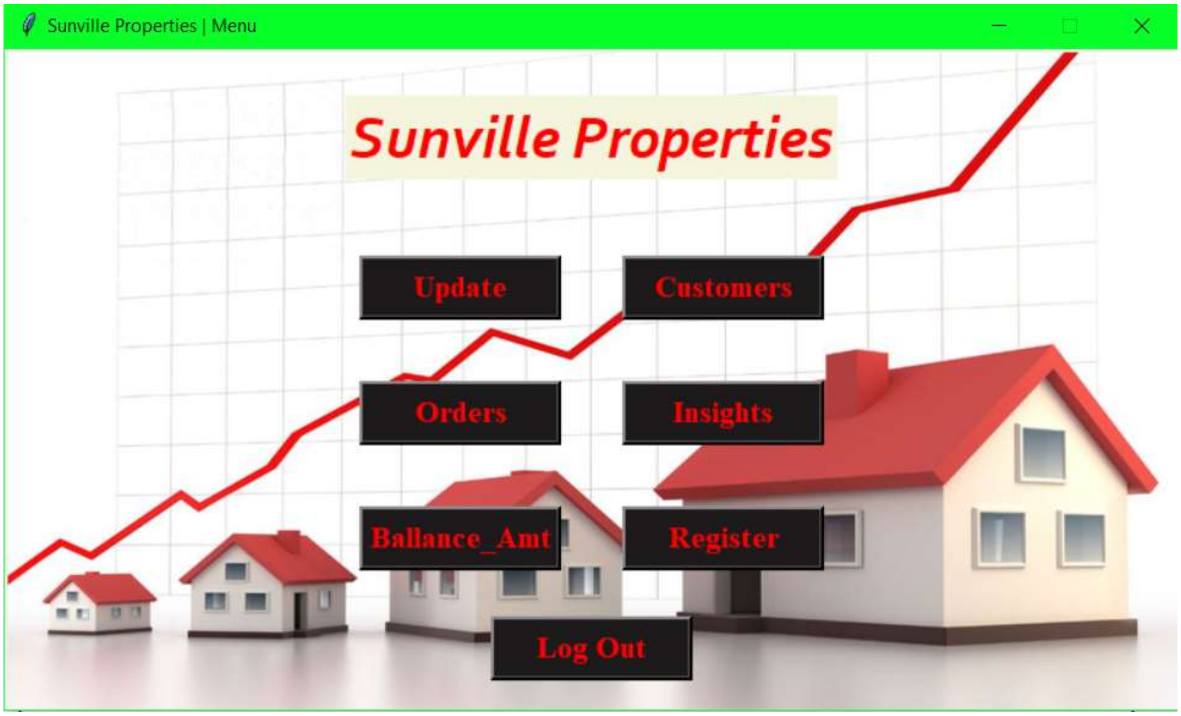
question: What is your mother's maiden name

Answer: Answer

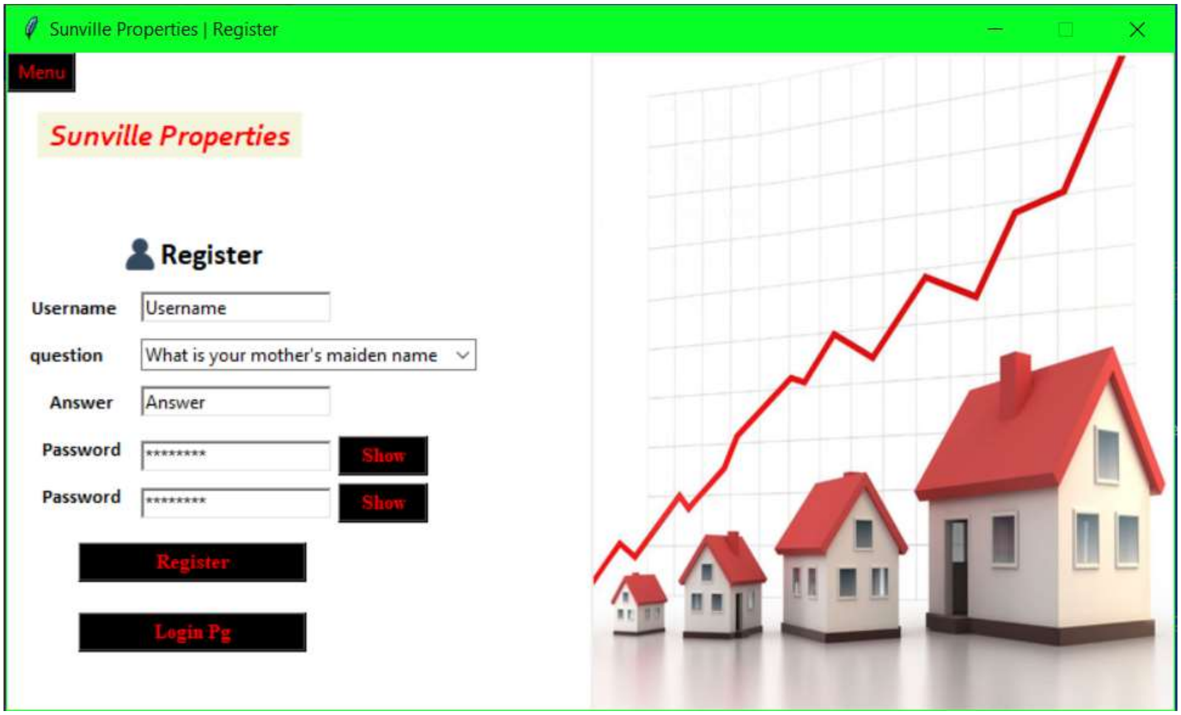
New Password: ********

New Password: ********

Menu page:



Register Page



Username: Username

question: What is your mother's maiden name

Answer: Answer

Password: ********

Password: ********

Update page:

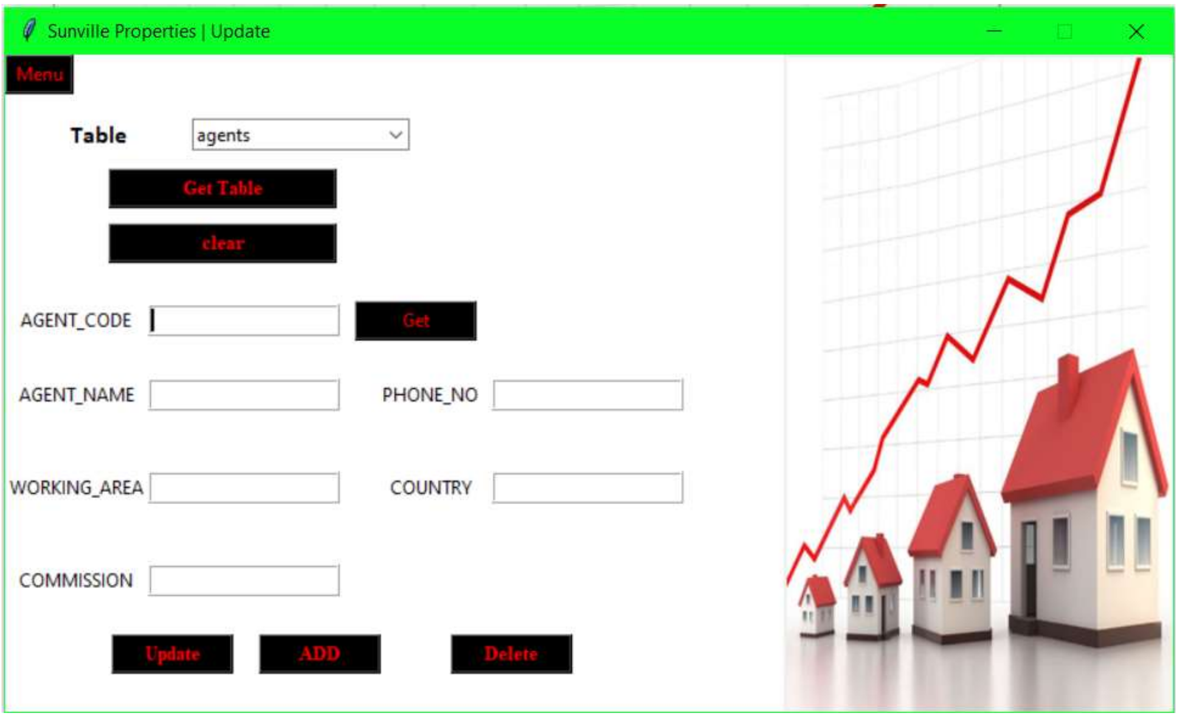
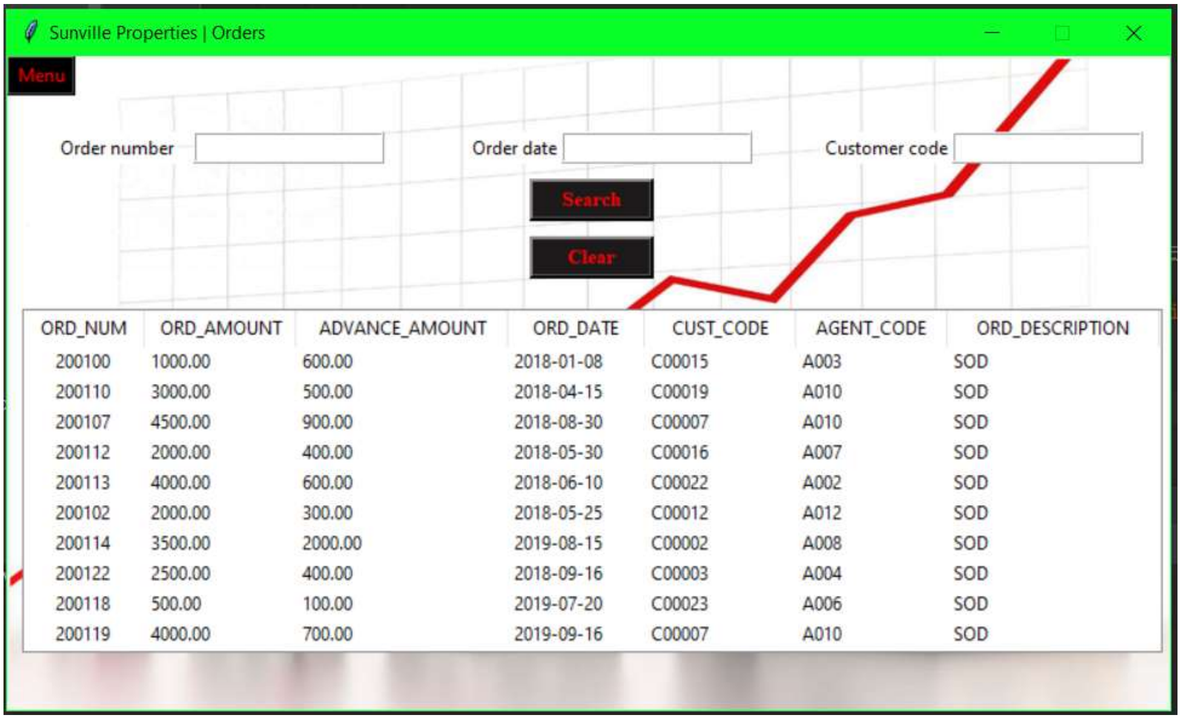


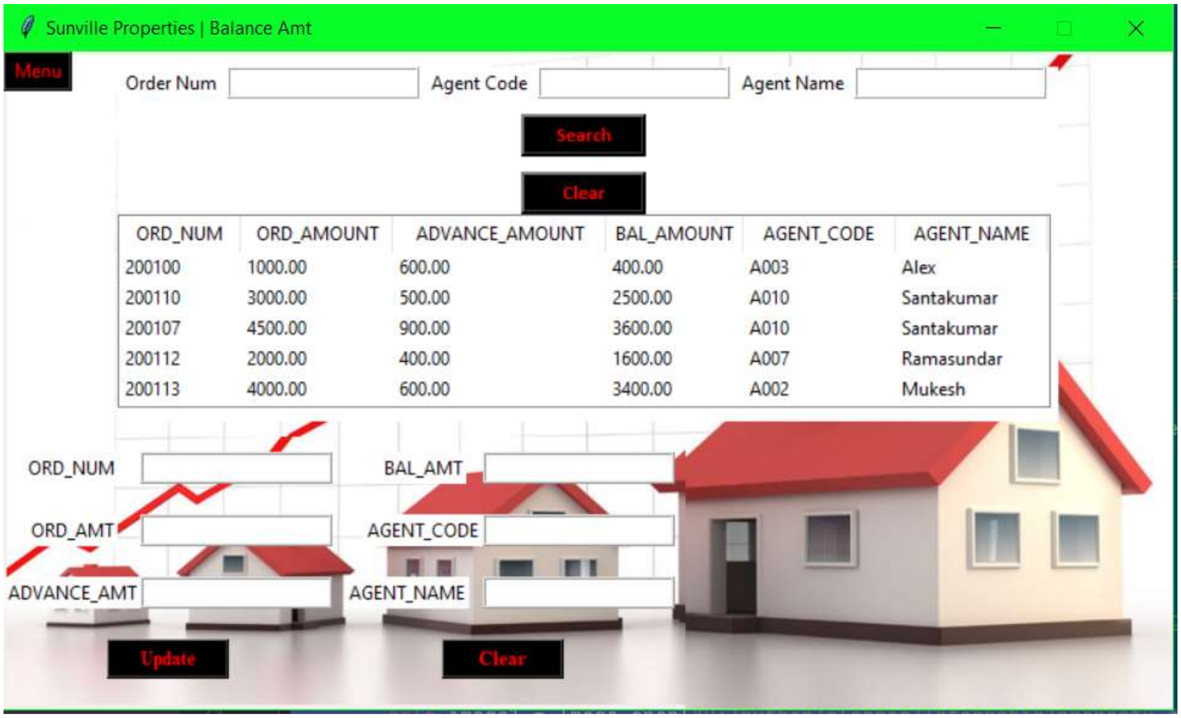
Table agents

Orders Page:



ORD_NUM	ORD_AMOUNT	ADVANCE_AMOUNT	ORD_DATE	CUST_CODE	AGENT_CODE	ORD_DESCRIPTION
200100	1000.00	600.00	2018-01-08	C00015	A003	SOD
200110	3000.00	500.00	2018-04-15	C00019	A010	SOD
200107	4500.00	900.00	2018-08-30	C00007	A010	SOD
200112	2000.00	400.00	2018-05-30	C00016	A007	SOD
200113	4000.00	600.00	2018-06-10	C00022	A002	SOD
200102	2000.00	300.00	2018-05-25	C00012	A012	SOD
200114	3500.00	2000.00	2019-08-15	C00002	A008	SOD
200122	2500.00	400.00	2018-09-16	C00003	A004	SOD
200118	500.00	100.00	2019-07-20	C00023	A006	SOD
200119	4000.00	700.00	2019-09-16	C00007	A010	SOD

Balance amount:



Order Num

Agent Code

Agent Name

ORD_NUM	ORD_AMOUNT	ADVANCE_AMOUNT	BAL_AMOUNT	AGENT_CODE	AGENT_NAME
200100	1000.00	600.00	400.00	A003	Alex
200110	3000.00	500.00	2500.00	A010	Santakumar
200107	4500.00	900.00	3600.00	A010	Santakumar
200112	2000.00	400.00	1600.00	A007	Ramasundar
200113	4000.00	600.00	3400.00	A002	Mukesh

ORD_NUM

BAL_AMT

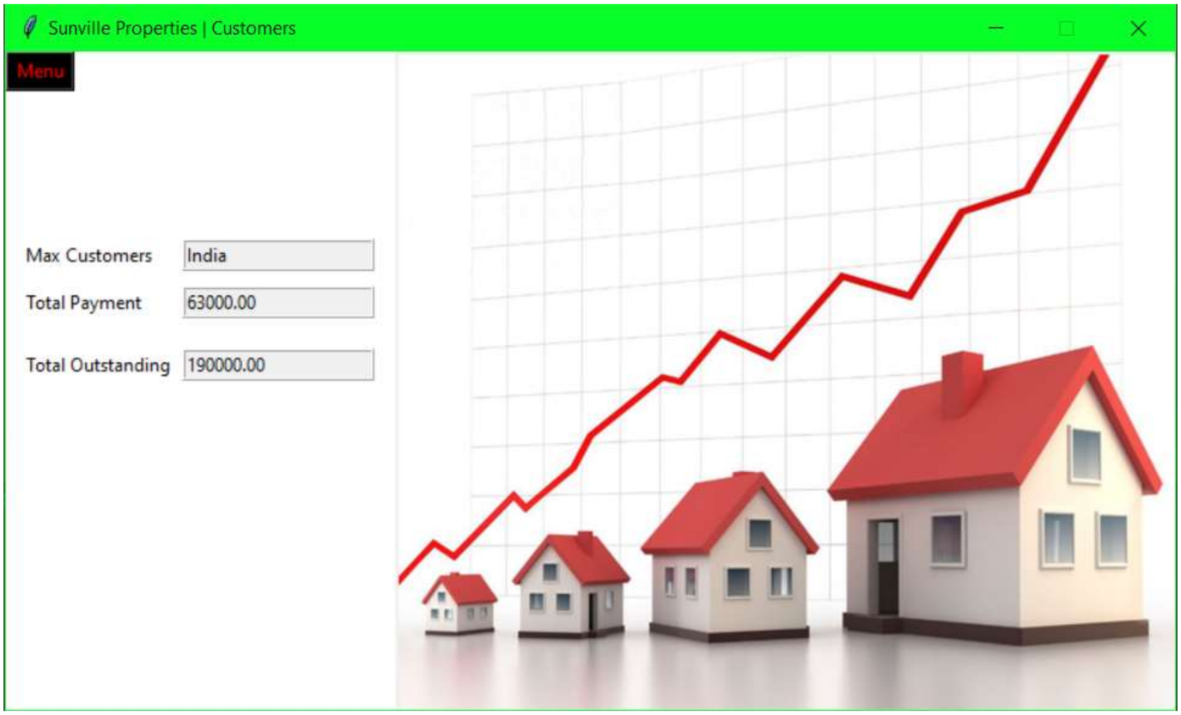
ORD_AMT

AGENT_CODE

ADVANCE_AMT

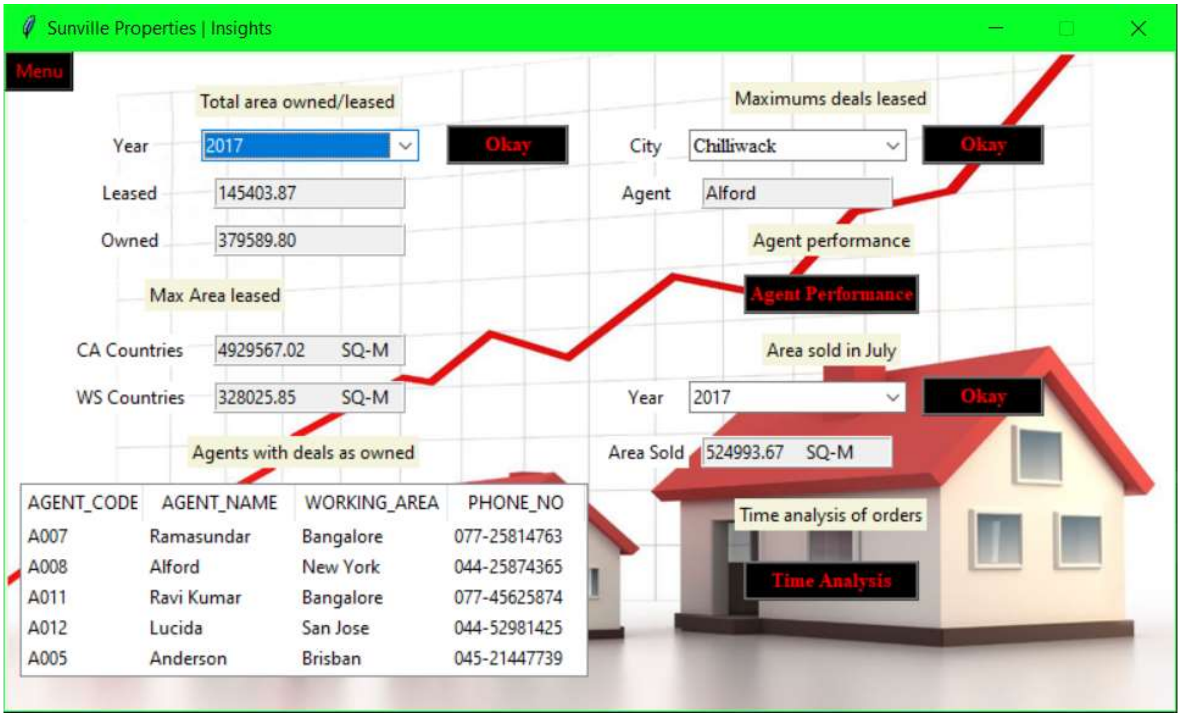
AGENT_NAME

Customers page:



Max Customers	India
Total Payment	63000.00
Total Outstanding	190000.00

Insights page:



Total area owned/leased

Year 2017

Leased 145403.87

Owned 379589.80

Max Area leased

CA Countries 4929567.02 SQ-M

WS Countries 328025.85 SQ-M

Maximums deals leased

City Chilliwack

Agent Alford

Agent performance

Area sold in July

Year 2017

Area Sold 524993.67 SQ-M

Time analysis of orders

Agents with deals as owned

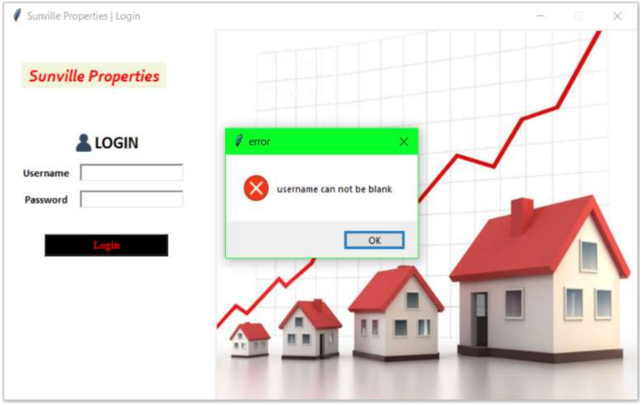
AGENT_CODE	AGENT_NAME	WORKING_AREA	PHONE_NO
A007	Ramasundar	Bangalore	077-25814763
A008	Alford	New York	044-25874365
A011	Ravi Kumar	Bangalore	077-45625874
A012	Lucida	San Jose	044-52981425
A005	Anderson	Brisban	045-21447739

Section 5:

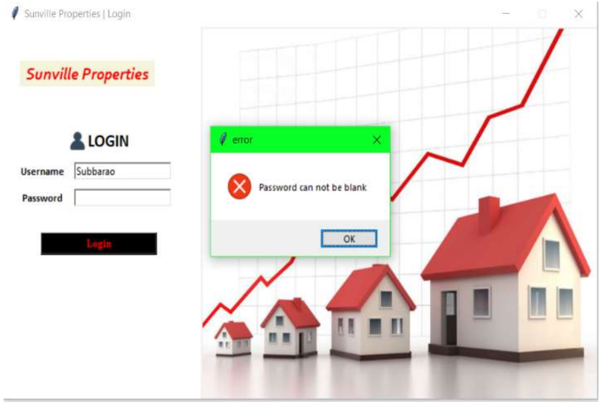
Testing:

The application has been tested to make it as fool-proof as possible. There are validations in place that do not allow wrong data format entry, as in entering letters where only numbers should be allowed or the correct phone no format and such

Login page

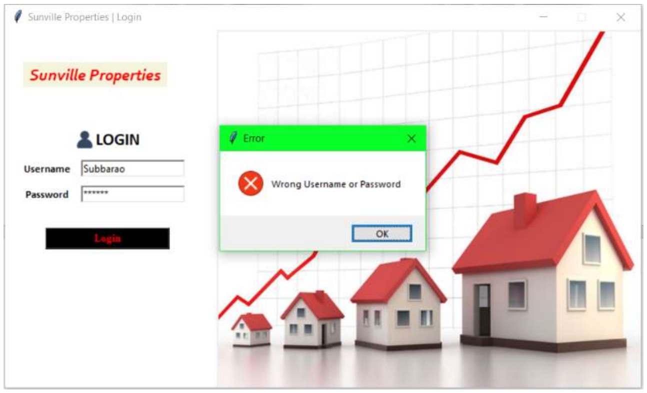


username can not be blank



Username: Subbarao

Password can not be blank

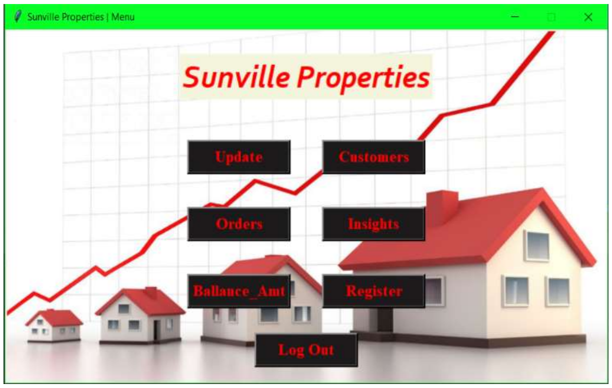


Username: Subbarao

Password: ******

Wrong Username or Password

On entering correct username and password



Update page:

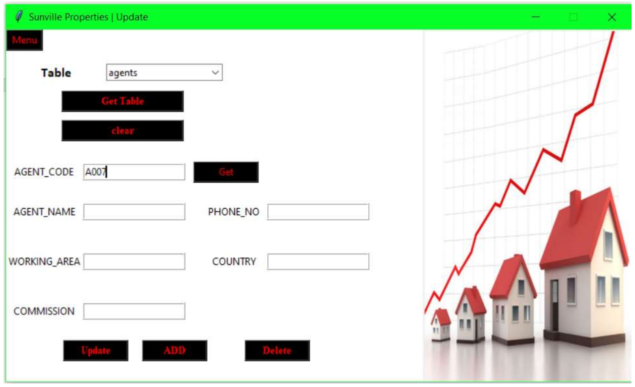


Table: agents

AGENT_CODE: A007

AGENT_NAME:

WORKING_AREA:

COMMISSION:

Using the get button:

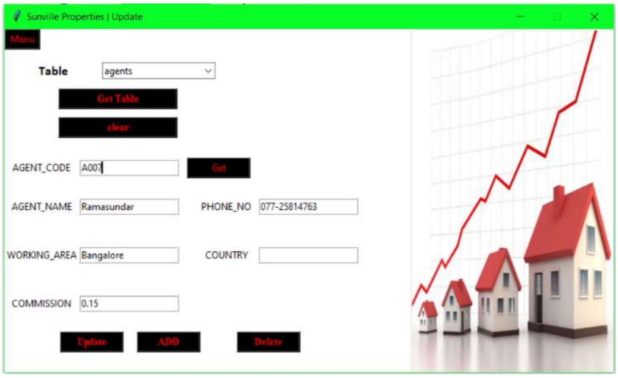


Table: agents

AGENT_CODE: A007

AGENT_NAME: Ramasundar

PHONE_NO: 077-25814763

WORKING_AREA: Bangalore

COUNTRY:

COMMISSION: 0.15

Using Add button:

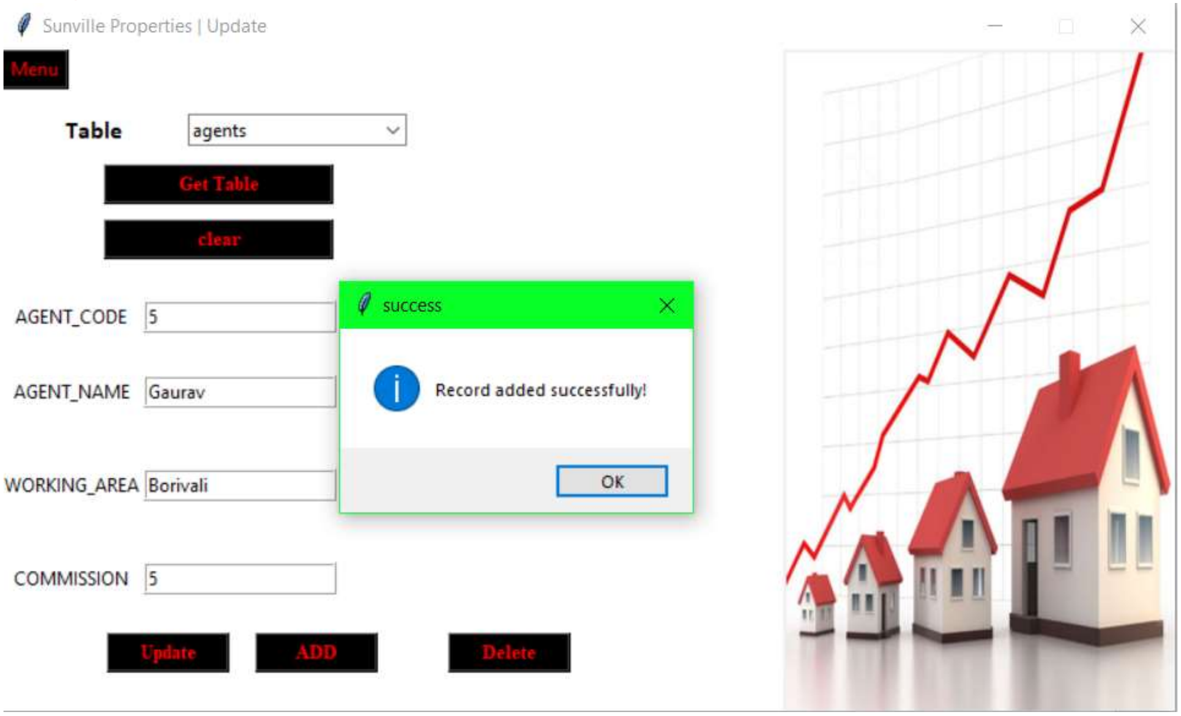


Table: agents

AGENT_CODE: 5

AGENT_NAME: Gaurav

WORKING_AREA: Borivali

COMMISSION: 5

Record added successfully!

When trying to add a record with code (primary key) that already exist It gives the option of weather to update the record or not

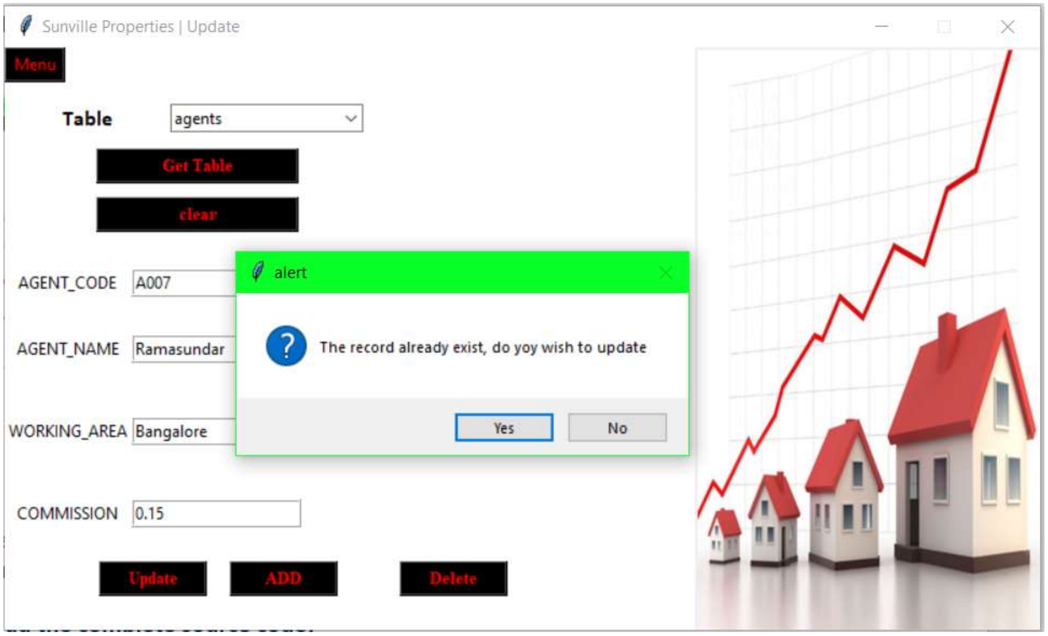


Table: agents

AGENT_CODE: A007

AGENT_NAME: Ramasundar

WORKING_AREA: Bangalore

COMMISSION: 0.15

The record already exist, do yoy wish to update

Using the get table button

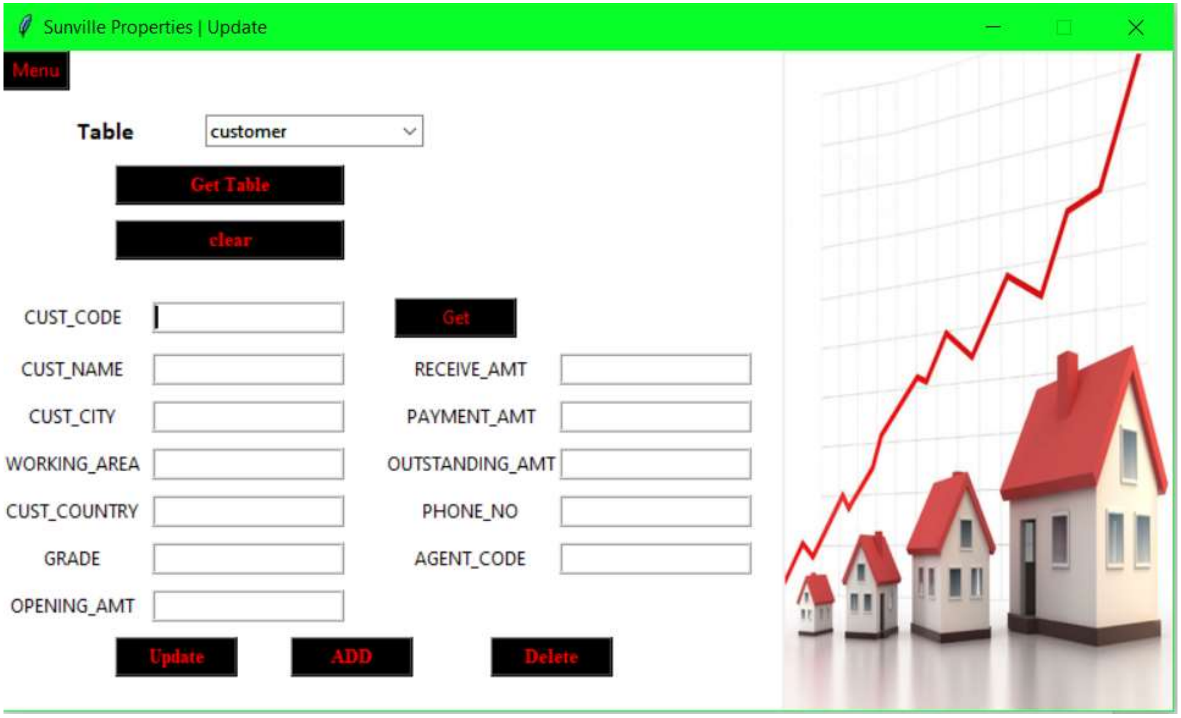


Table: customer

CUST_CODE
CUST_NAME
CUST_CITY
WORKING_AREA
CUST_COUNTRY
GRADE
OPENING_AMT
RECEIVE_AMT
PAYMENT_AMT
OUTSTANDING_AMT
PHONE_NO
AGENT_CODE

Using the clear button:

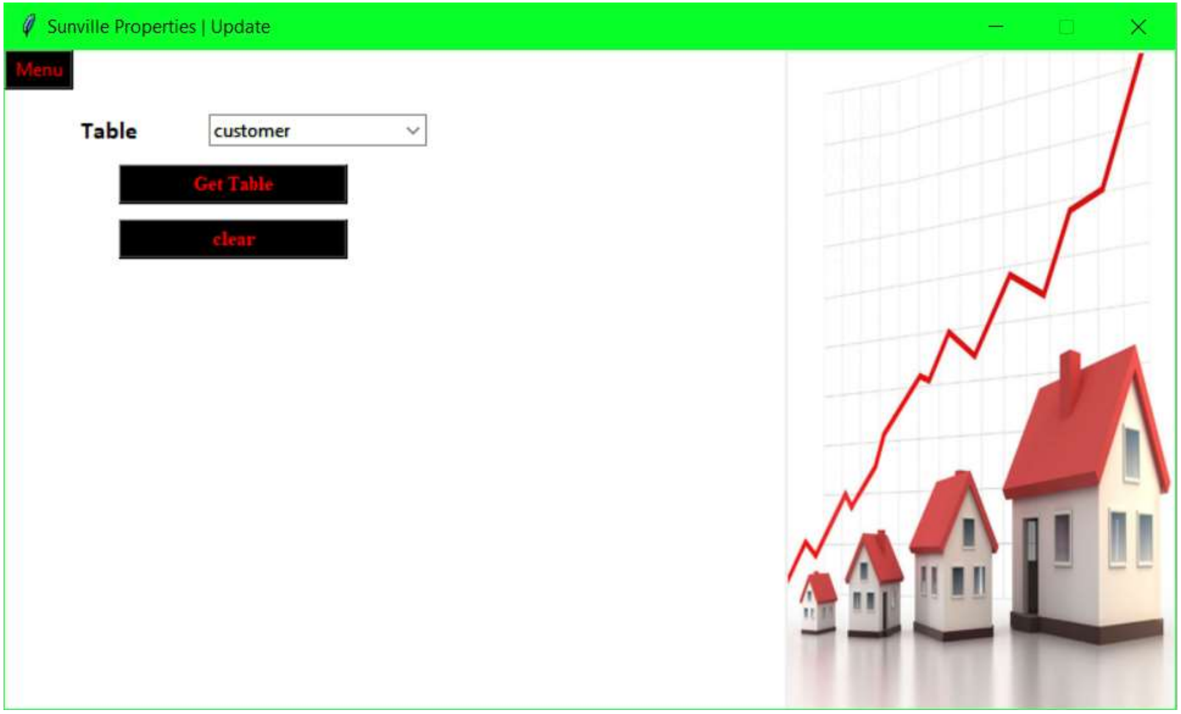


Table customer

Using update button:

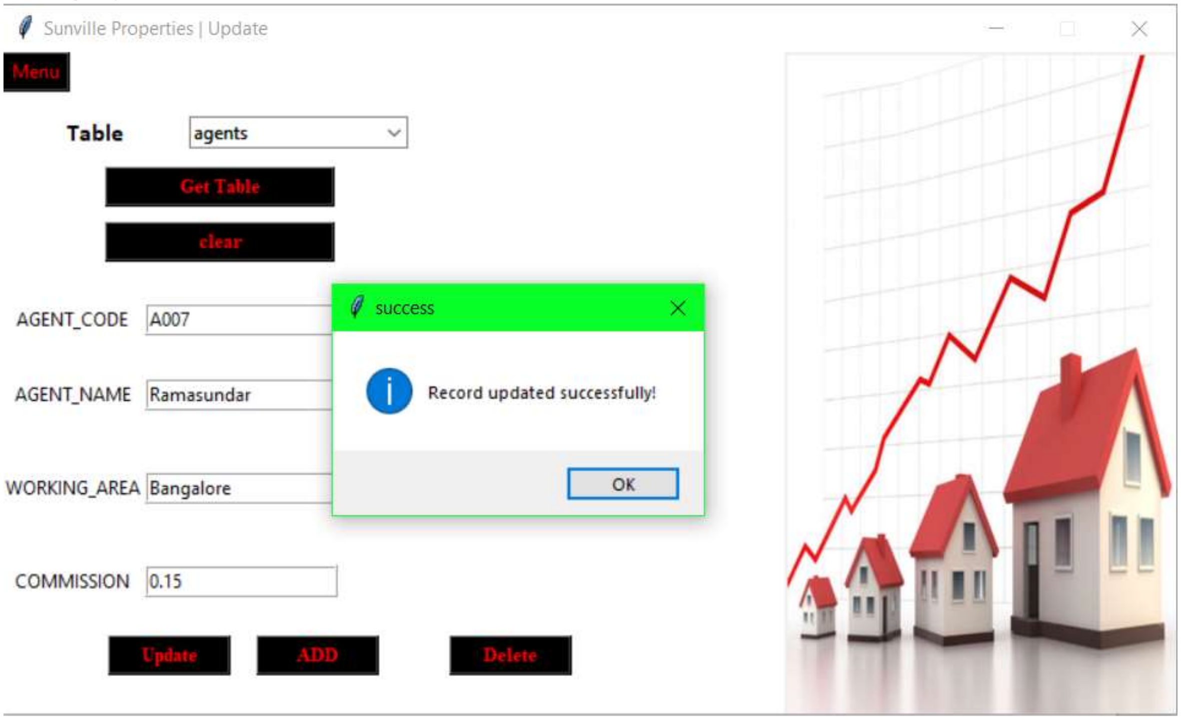


Table agents

AGENT_CODE A007

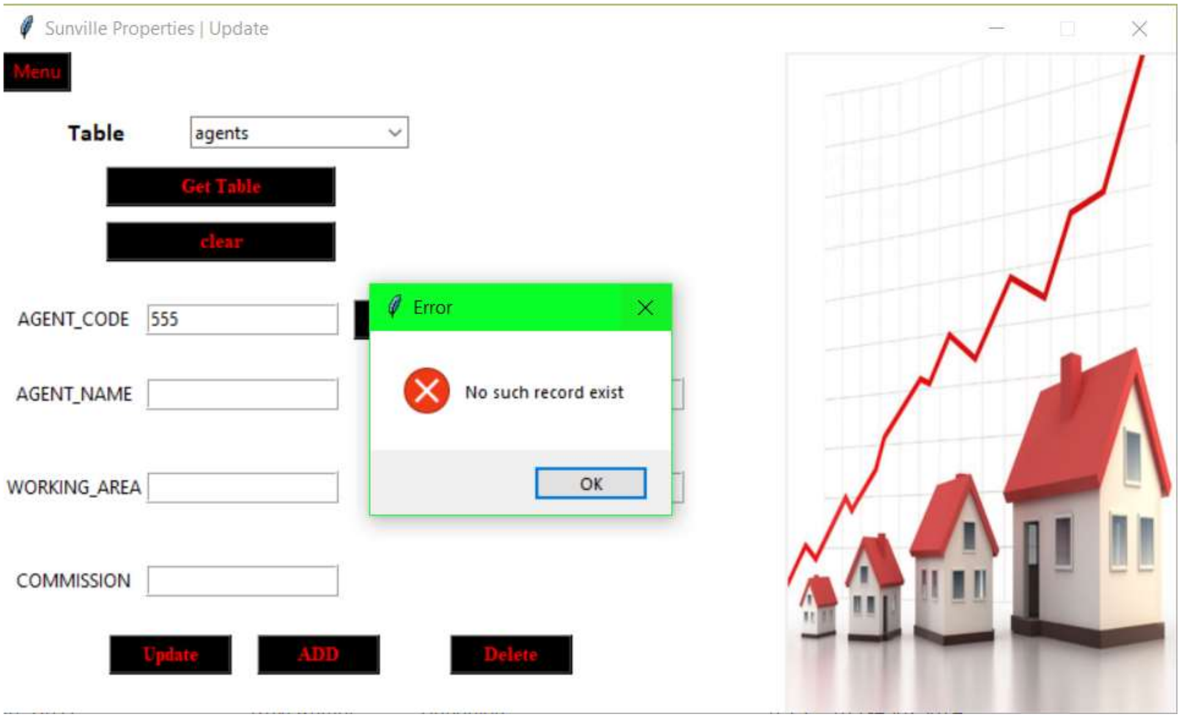
AGENT_NAME Ramasundar

WORKING_AREA Bangalore

COMMISSION 0.15

Record updated successfully!

Using get button on a record that doesn't exist:



Sunville Properties | Update

Table: agents

AGENT_CODE: 555

AGENT_NAME:

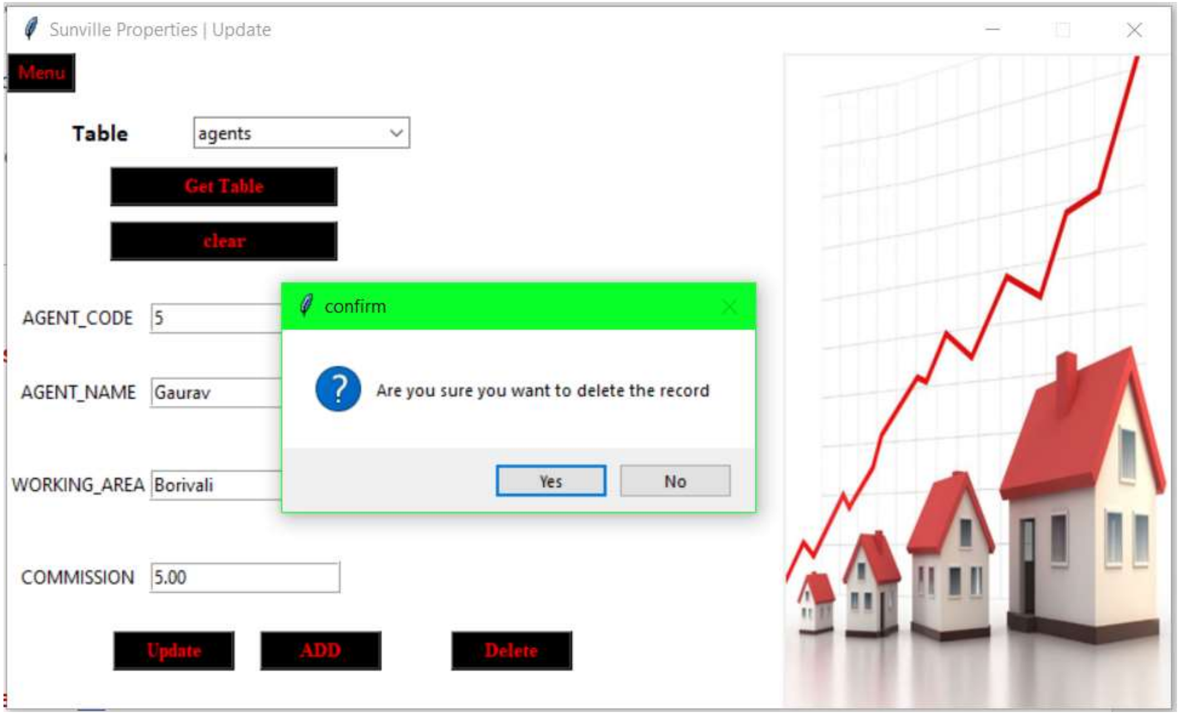
WORKING_AREA:

COMMISSION:

Error

No such record exist

Using the delete button:



Sunville Properties | Update

Table: agents

AGENT_CODE: 5

AGENT_NAME: Gaurav

WORKING_AREA: Borivali

COMMISSION: 5.00

confirm

Are you sure you want to delete the record

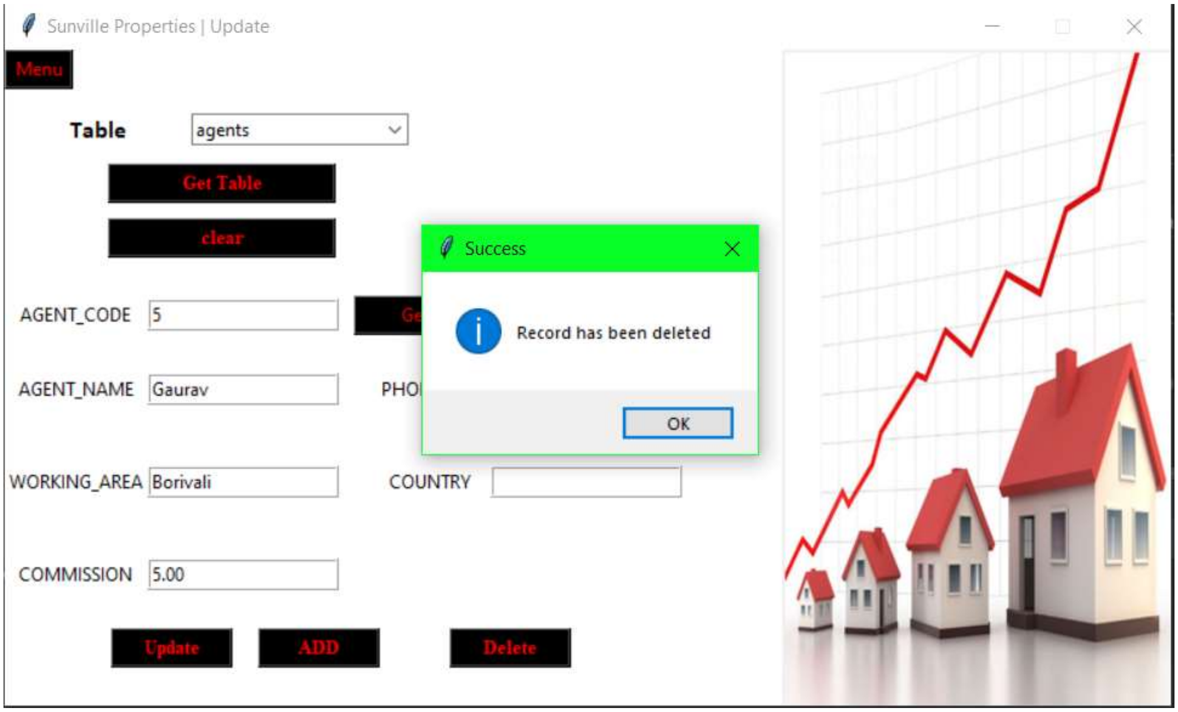


Table: agents

AGENT_CODE: 5

AGENT_NAME: Gaurav

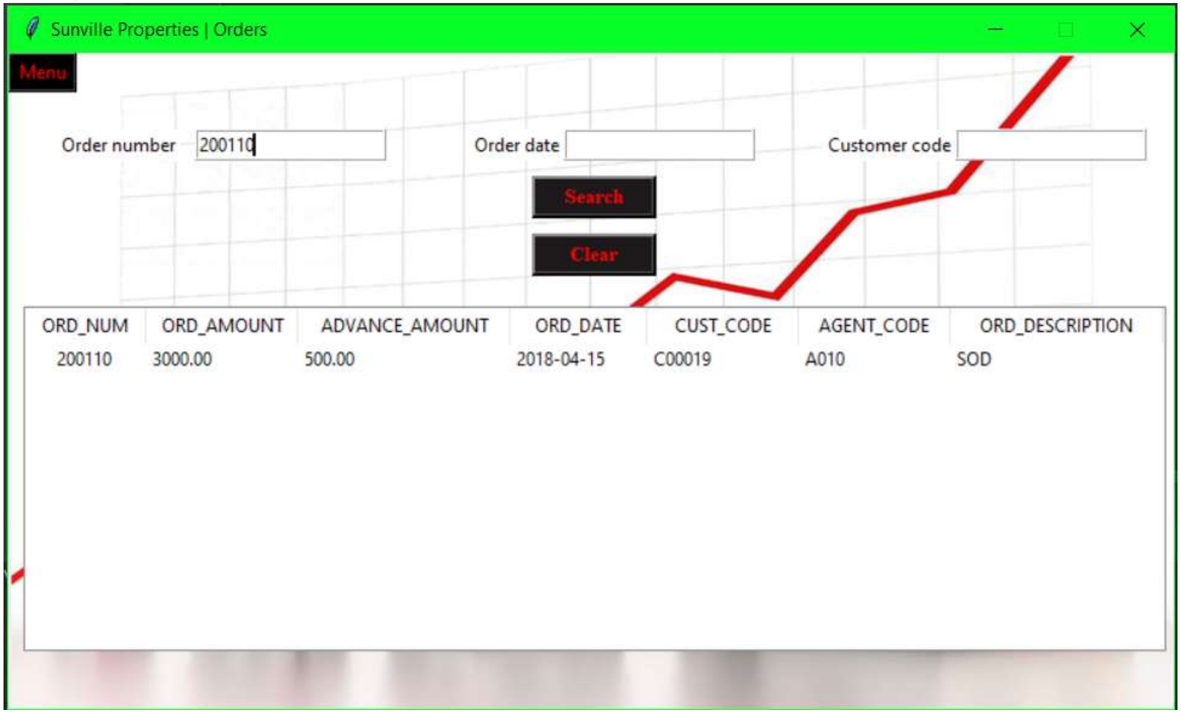
WORKING_AREA: Borivali

COUNTRY:

COMMISSION: 5.00

Record has been deleted

Orders page:
Using search with one field:

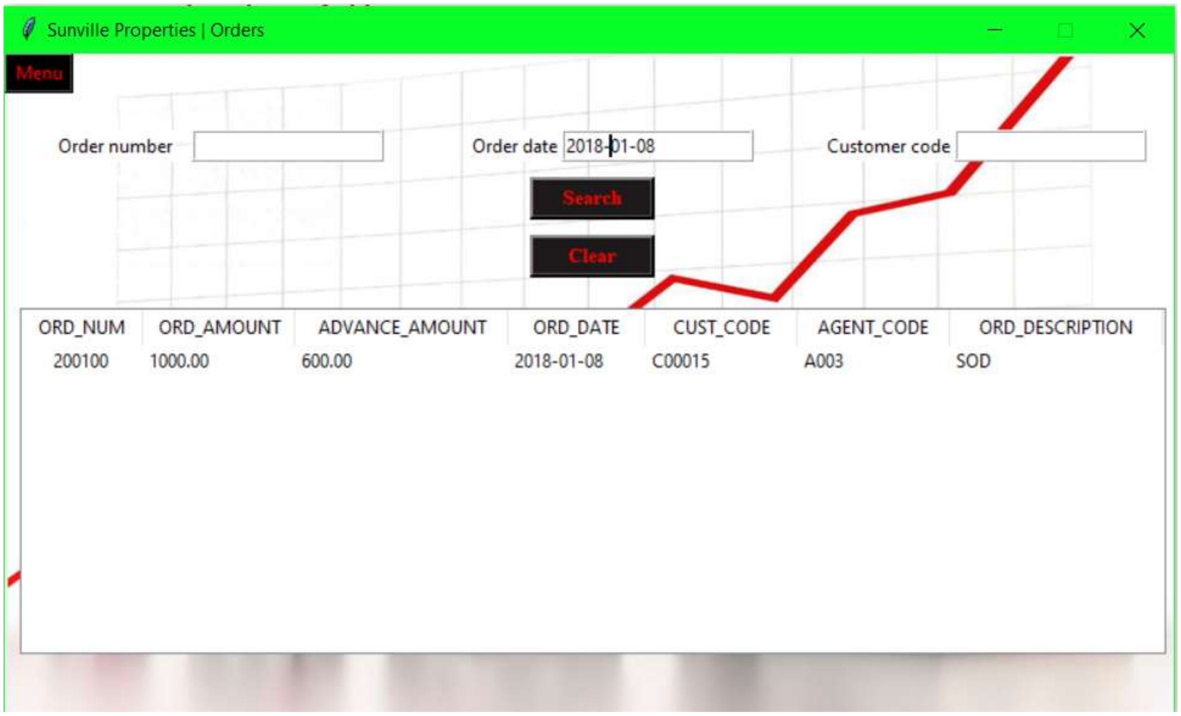


Order number: 200110

Order date:

Customer code:

ORD_NUM	ORD_AMOUNT	ADVANCE_AMOUNT	ORD_DATE	CUST_CODE	AGENT_CODE	ORD_DESCRIPTION
200110	3000.00	500.00	2018-04-15	C00019	A010	SOD

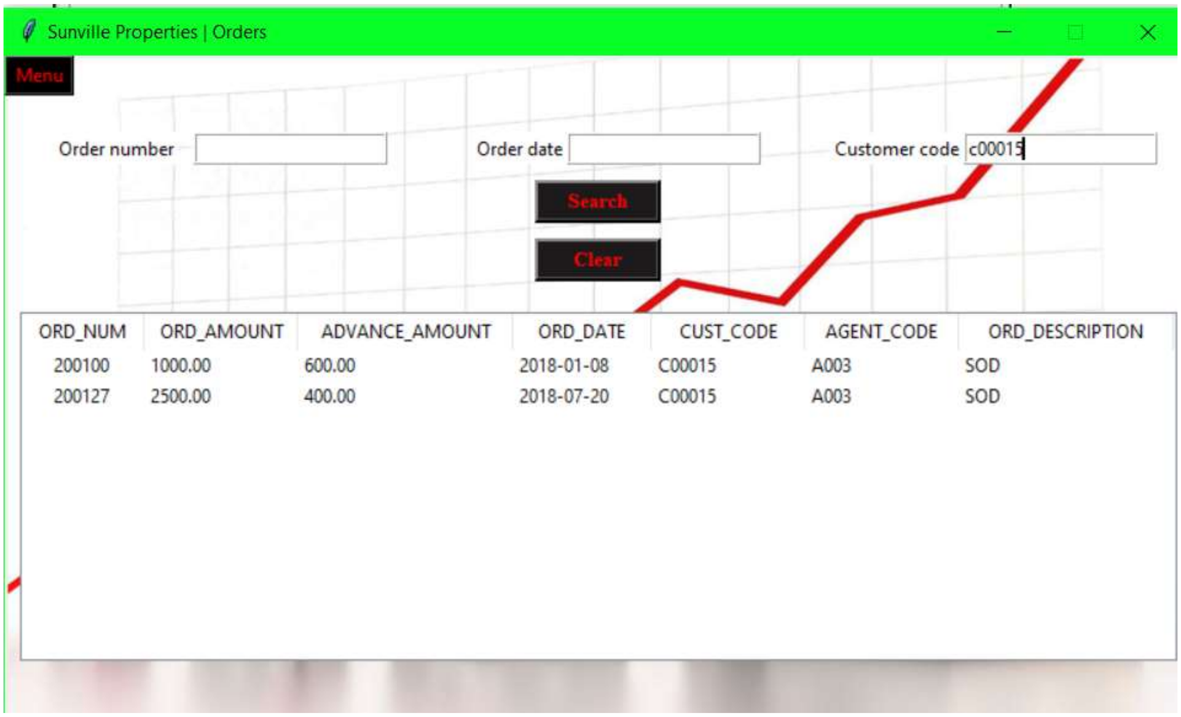


Order number

Order date 2018-01-08

Customer code

ORD_NUM	ORD_AMOUNT	ADVANCE_AMOUNT	ORD_DATE	CUST_CODE	AGENT_CODE	ORD_DESCRIPTION
200100	1000.00	600.00	2018-01-08	C00015	A003	SOD



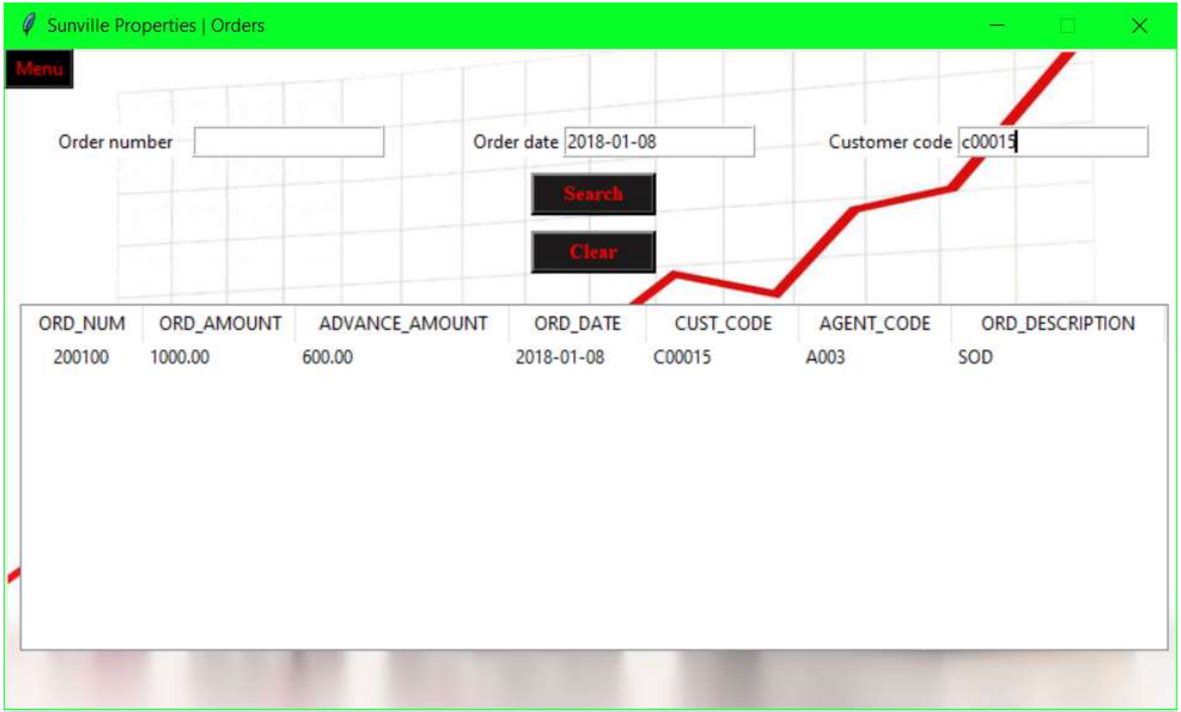
Order number

Order date

Customer code c00015

ORD_NUM	ORD_AMOUNT	ADVANCE_AMOUNT	ORD_DATE	CUST_CODE	AGENT_CODE	ORD_DESCRIPTION
200100	1000.00	600.00	2018-01-08	C00015	A003	SOD
200127	2500.00	400.00	2018-07-20	C00015	A003	SOD

Using 2 fields:

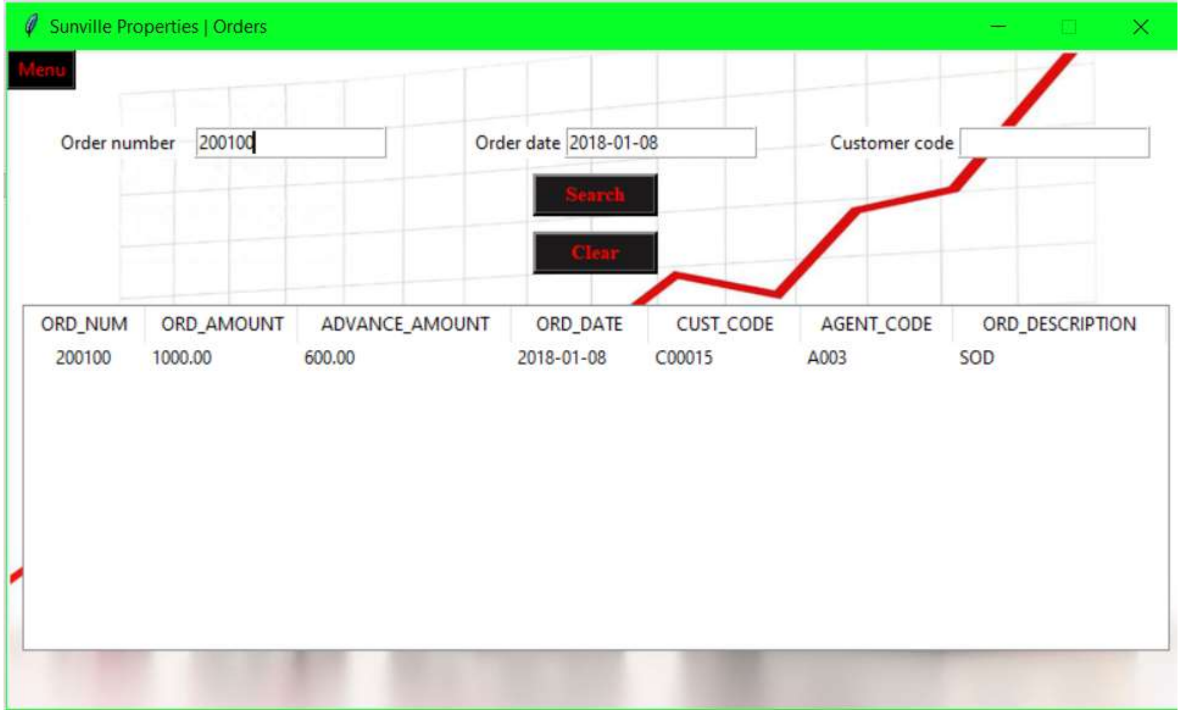


Order number

Order date 2018-01-08

Customer code c00015

ORD_NUM	ORD_AMOUNT	ADVANCE_AMOUNT	ORD_DATE	CUST_CODE	AGENT_CODE	ORD_DESCRIPTION
200100	1000.00	600.00	2018-01-08	C00015	A003	SOD

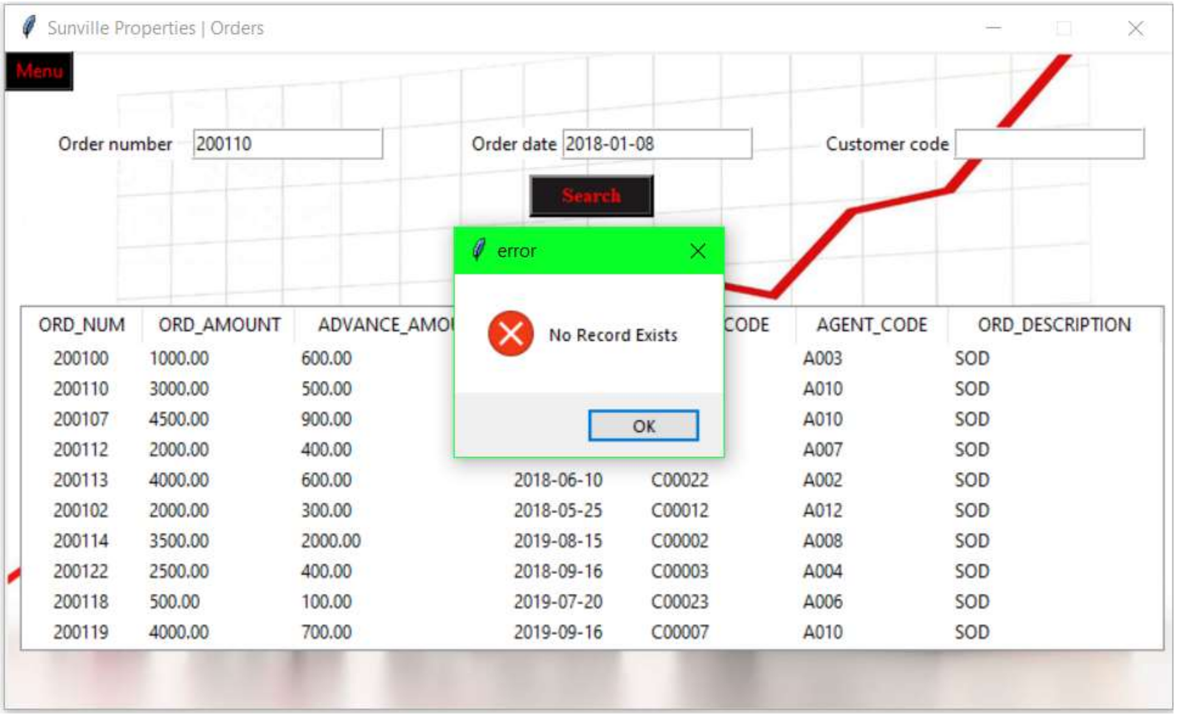


Order number 200100

Order date 2018-01-08

Customer code

ORD_NUM	ORD_AMOUNT	ADVANCE_AMOUNT	ORD_DATE	CUST_CODE	AGENT_CODE	ORD_DESCRIPTION
200100	1000.00	600.00	2018-01-08	C00015	A003	SOD



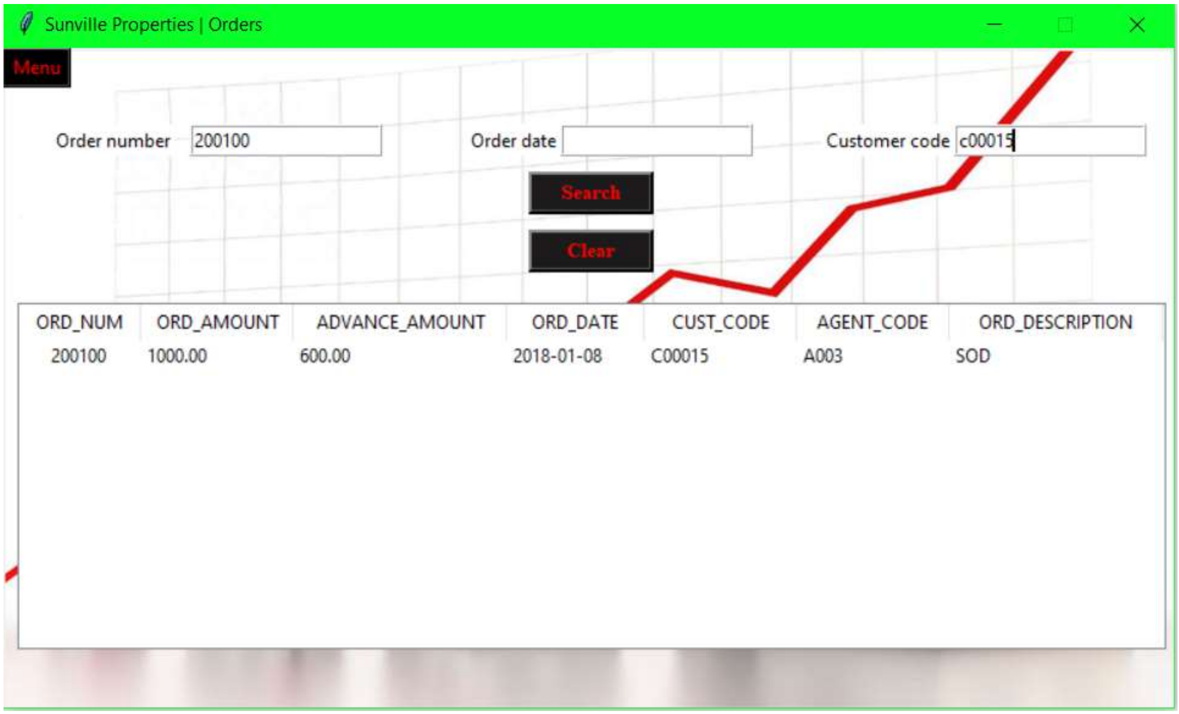
Order number 200110

Order date 2018-01-08

Customer code

No Record Exists

ORD_NUM	ORD_AMOUNT	ADVANCE_AMOU		CODE	AGENT_CODE	ORD_DESCRIPTION
200100	1000.00	600.00			A003	SOD
200110	3000.00	500.00			A010	SOD
200107	4500.00	900.00			A010	SOD
200112	2000.00	400.00			A007	SOD
200113	4000.00	600.00	2018-06-10	C00022	A002	SOD
200102	2000.00	300.00	2018-05-25	C00012	A012	SOD
200114	3500.00	2000.00	2019-08-15	C00002	A008	SOD
200122	2500.00	400.00	2018-09-16	C00003	A004	SOD
200118	500.00	100.00	2019-07-20	C00023	A006	SOD
200119	4000.00	700.00	2019-09-16	C00007	A010	SOD



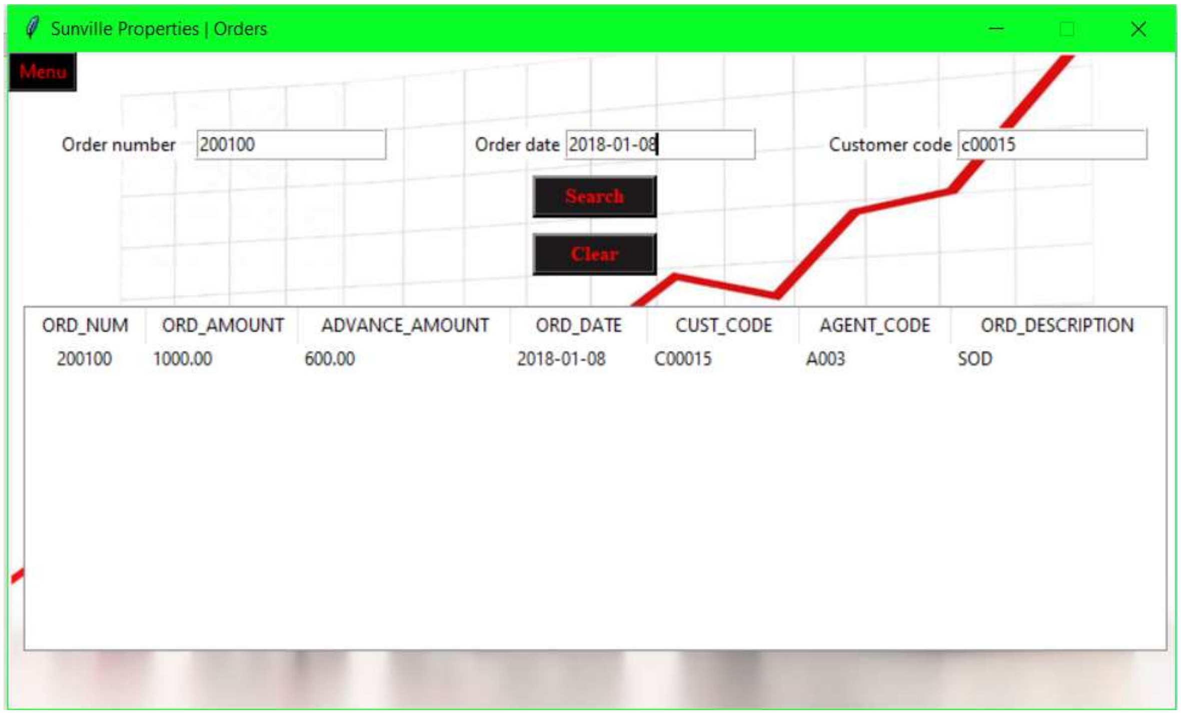
Order number 200100

Order date

Customer code c00015

ORD_NUM	ORD_AMOUNT	ADVANCE_AMOUNT	ORD_DATE	CUST_CODE	AGENT_CODE	ORD_DESCRIPTION
200100	1000.00	600.00	2018-01-08	C00015	A003	SOD

All 3 fields:



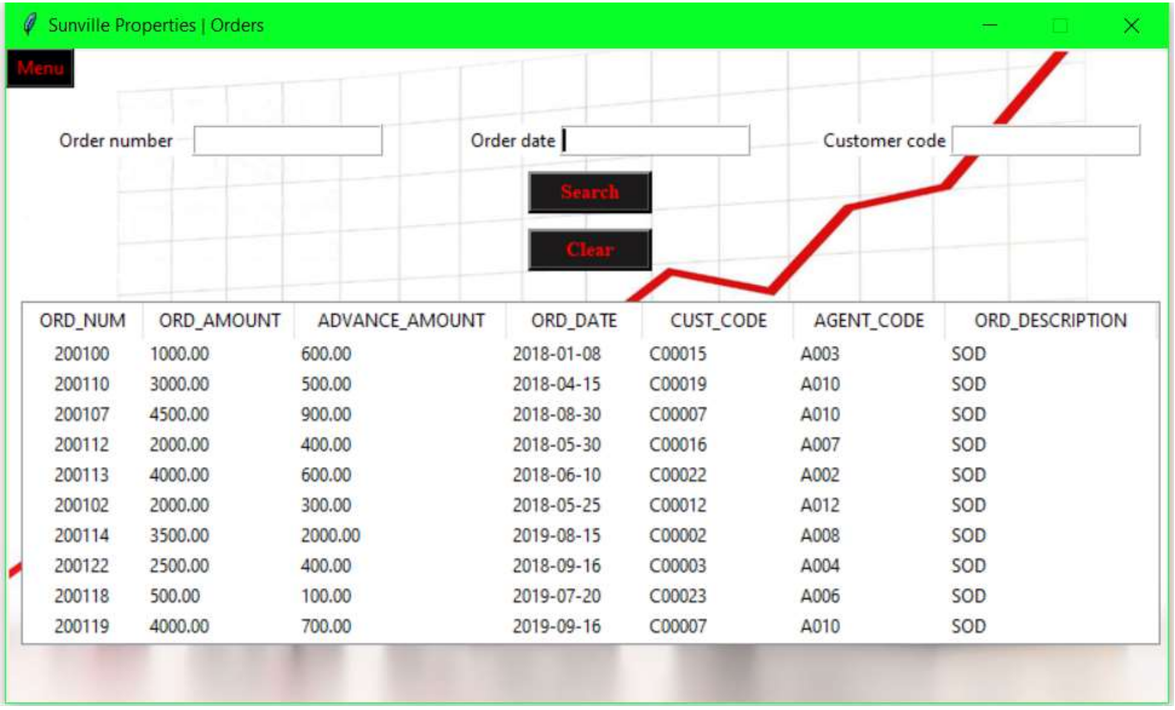
Order number 200100

Order date 2018-01-08

Customer code c00015

ORD_NUM	ORD_AMOUNT	ADVANCE_AMOUNT	ORD_DATE	CUST_CODE	AGENT_CODE	ORD_DESCRIPTION
200100	1000.00	600.00	2018-01-08	C00015	A003	SOD

Using 'clear' button:



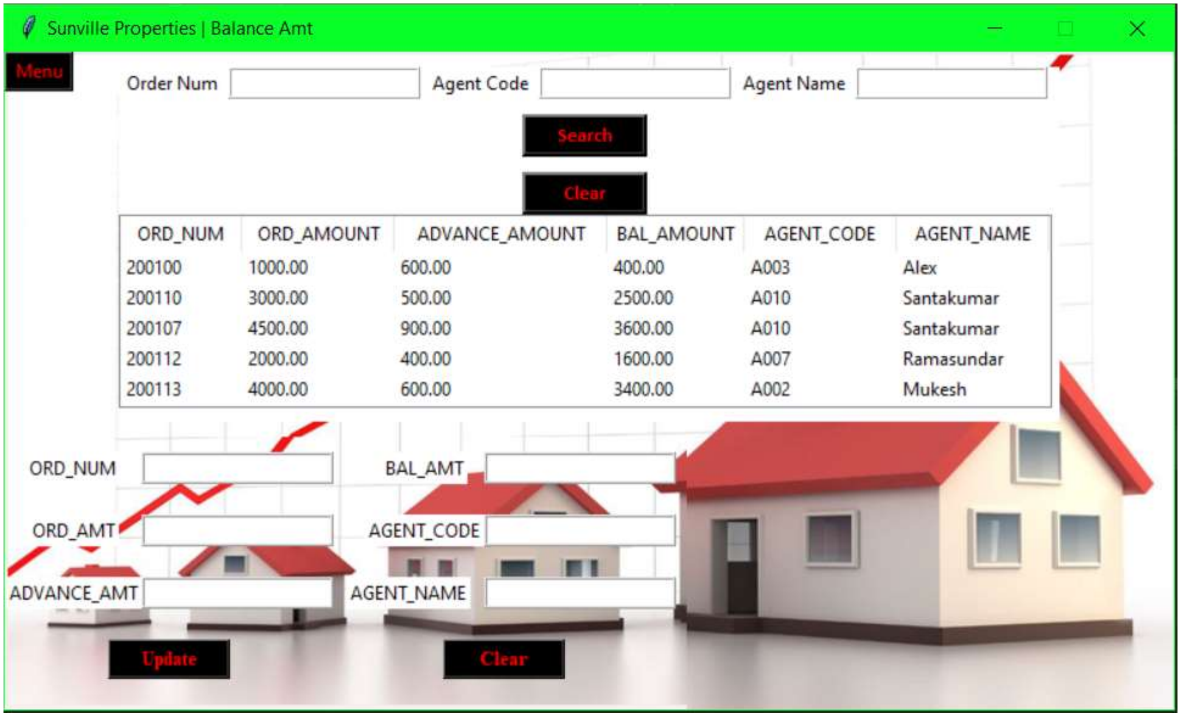
Order number

Order date

Customer code

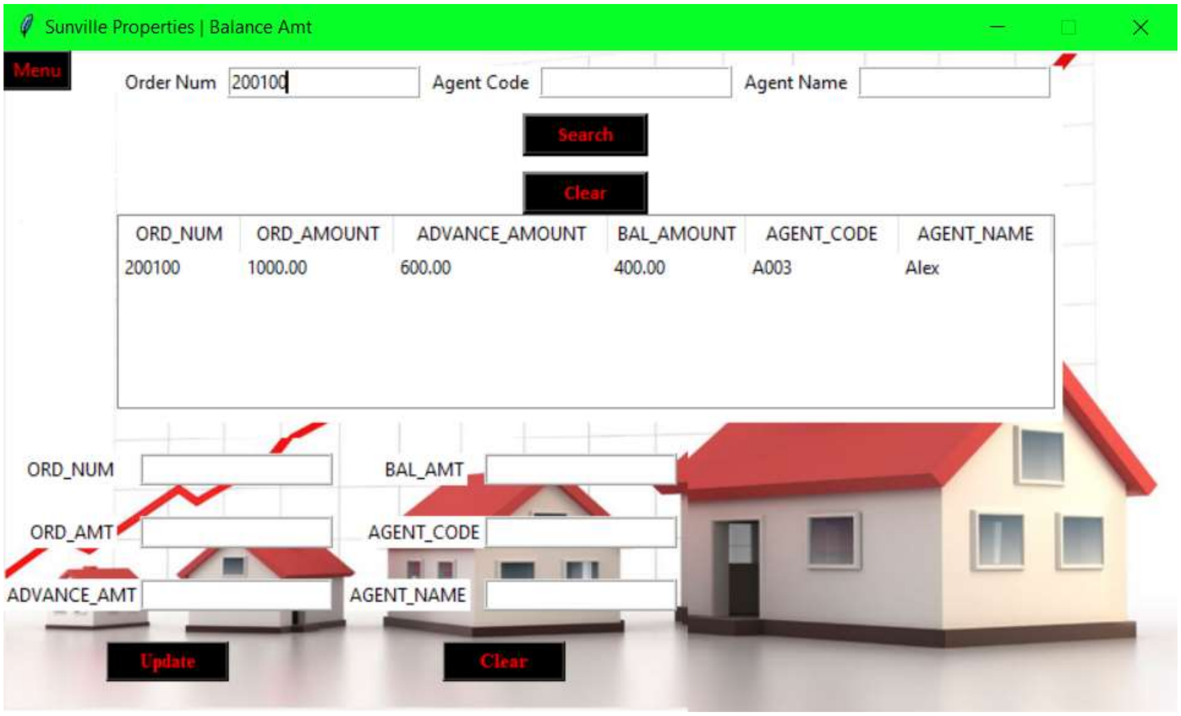
ORD_NUM	ORD_AMOUNT	ADVANCE_AMOUNT	ORD_DATE	CUST_CODE	AGENT_CODE	ORD_DESCRIPTION
200100	1000.00	600.00	2018-01-08	C00015	A003	SOD
200110	3000.00	500.00	2018-04-15	C00019	A010	SOD
200107	4500.00	900.00	2018-08-30	C00007	A010	SOD
200112	2000.00	400.00	2018-05-30	C00016	A007	SOD
200113	4000.00	600.00	2018-06-10	C00022	A002	SOD
200102	2000.00	300.00	2018-05-25	C00012	A012	SOD
200114	3500.00	2000.00	2019-08-15	C00002	A008	SOD
200122	2500.00	400.00	2018-09-16	C00003	A004	SOD
200118	500.00	100.00	2019-07-20	C00023	A006	SOD
200119	4000.00	700.00	2019-09-16	C00007	A010	SOD

Balance amount page:



ORD_NUM	ORD_AMOUNT	ADVANCE_AMOUNT	BAL_AMOUNT	AGENT_CODE	AGENT_NAME
200100	1000.00	600.00	400.00	A003	Alex
200110	3000.00	500.00	2500.00	A010	Santakumar
200107	4500.00	900.00	3600.00	A010	Santakumar
200112	2000.00	400.00	1600.00	A007	Ramasundar
200113	4000.00	600.00	3400.00	A002	Mukesh

Search fields:



Order Num 200100

Agent Code

Agent Name

ORD_NUM	ORD_AMOUNT	ADVANCE_AMOUNT	BAL_AMOUNT	AGENT_CODE	AGENT_NAME
200100	1000.00	600.00	400.00	A003	Alex

ORD_NUM

ORD_AMT

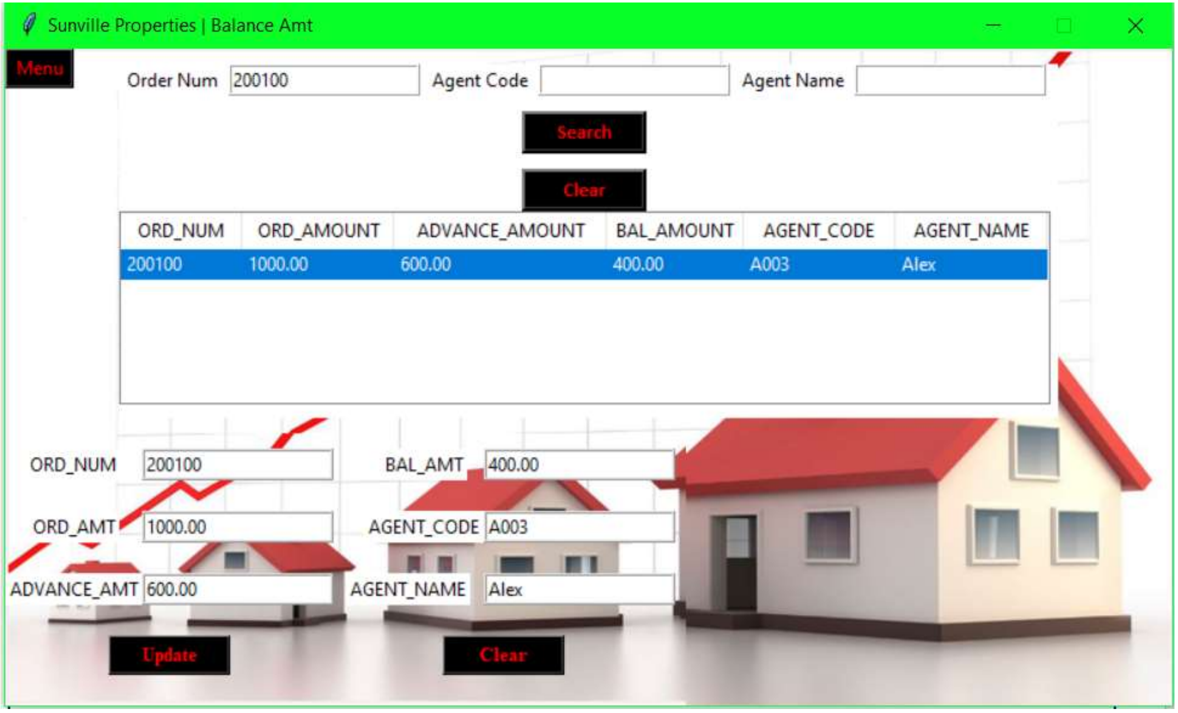
ADVANCE_AMT

BAL_AMT

AGENT_CODE

AGENT_NAME

double click on a record to get value:



Order Num 200100

Agent Code

Agent Name

ORD_NUM	ORD_AMOUNT	ADVANCE_AMOUNT	BAL_AMOUNT	AGENT_CODE	AGENT_NAME
200100	1000.00	600.00	400.00	A003	Alex

ORD_NUM 200100

ORD_AMT 1000.00

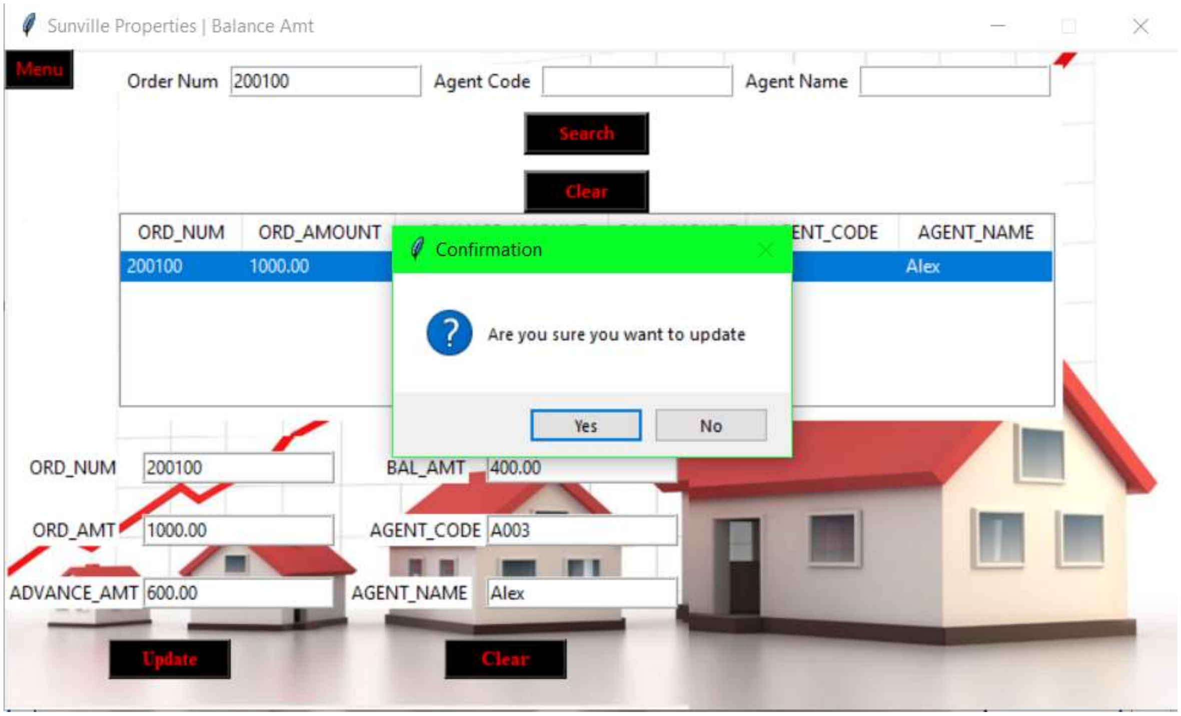
ADVANCE_AMT 600.00

BAL_AMT 400.00

AGENT_CODE A003

AGENT_NAME Alex

Update button:



Order Num 200100 Agent Code Agent Name

ORD_NUM	ORD_AMOUNT	ENT_CODE	AGENT_NAME
200100	1000.00		Alex

Are you sure you want to update

ORD_NUM 200100

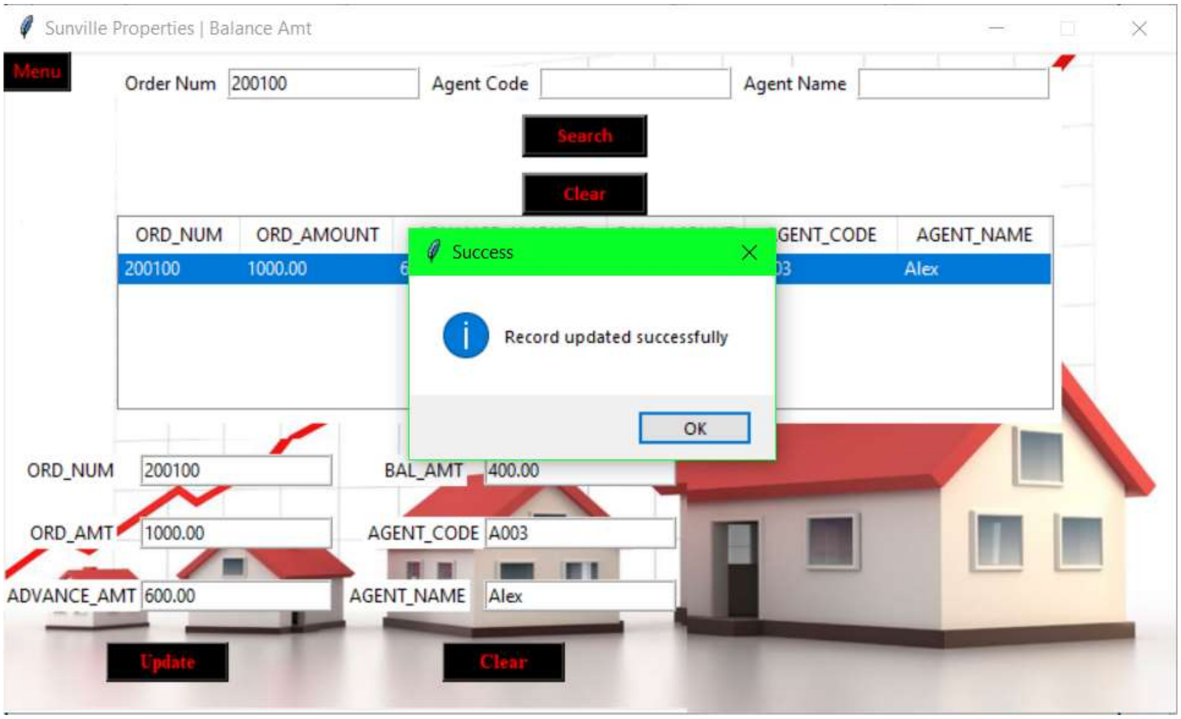
BAL_AMT 400.00

ORD_AMT 1000.00

AGENT_CODE A003

ADVANCE_AMT 600.00

AGENT_NAME Alex



Order Num 200100 Agent Code Agent Name

ORD_NUM	ORD_AMOUNT	GENT_CODE	AGENT_NAME
200100	1000.00 6	3	Alex

Record updated successfully

ORD_NUM 200100

BAL_AMT 400.00

ORD_AMT 1000.00

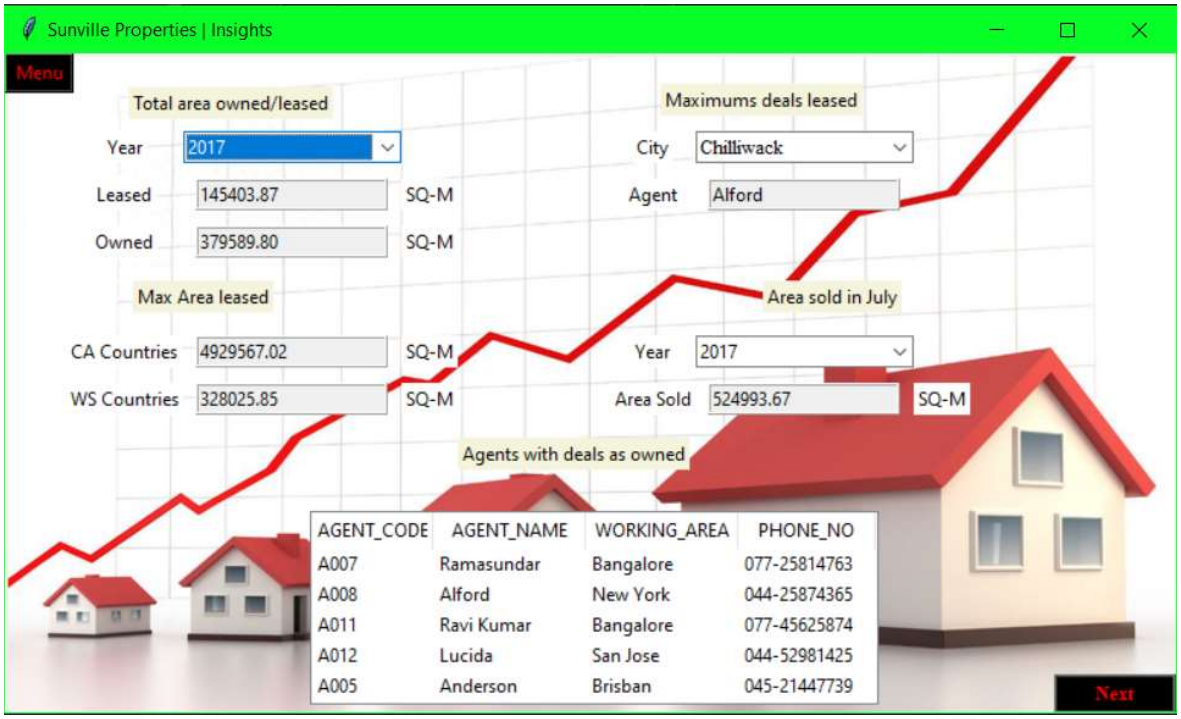
AGENT_CODE A003

ADVANCE_AMT 600.00

AGENT_NAME Alex

Insights page:

Year is changed in ‘total area owned/leased’ section:



Total area owned/leased

Year 2017

Leased 145403.87 SQ-M

Owned 379589.80 SQ-M

Max Area leased

CA Countries 4929567.02 SQ-M

WS Countries 328025.85 SQ-M

Maximums deals leased

City Chilliwack

Agent Alford

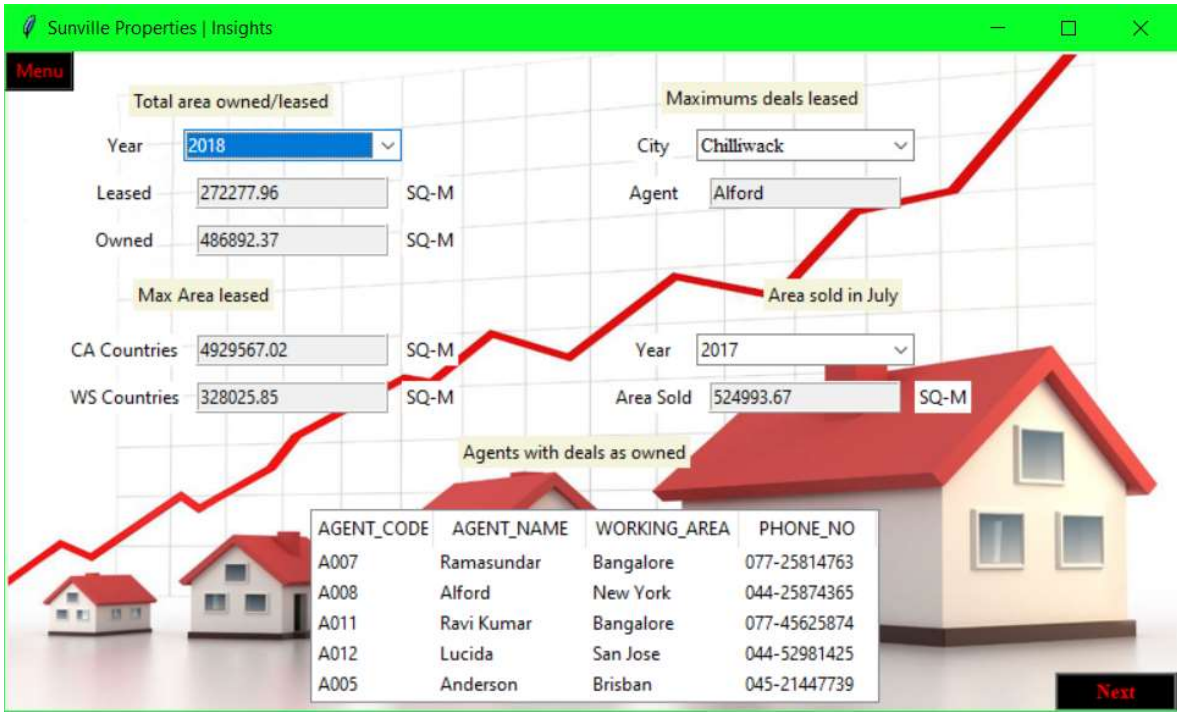
Area sold in July

Year 2017

Area Sold 524993.67 SQ-M

Agents with deals as owned

AGENT_CODE	AGENT_NAME	WORKING_AREA	PHONE_NO
A007	Ramasundar	Bangalore	077-25814763
A008	Alford	New York	044-25874365
A011	Ravi Kumar	Bangalore	077-45625874
A012	Lucida	San Jose	044-52981425
A005	Anderson	Brisban	045-21447739



Total area owned/leased

Year 2018

Leased 272277.96 SQ-M

Owned 486892.37 SQ-M

Max Area leased

CA Countries 4929567.02 SQ-M

WS Countries 328025.85 SQ-M

Maximums deals leased

City Chilliwack

Agent Alford

Area sold in July

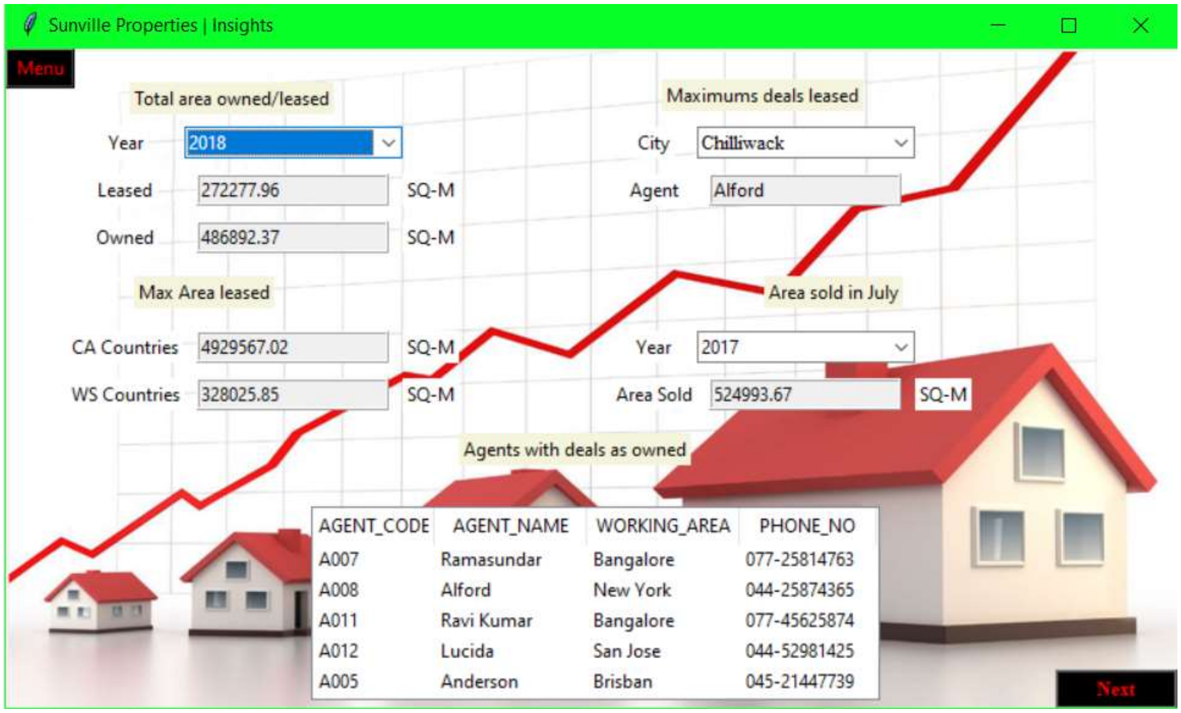
Year 2017

Area Sold 524993.67 SQ-M

Agents with deals as owned

AGENT_CODE	AGENT_NAME	WORKING_AREA	PHONE_NO
A007	Ramasundar	Bangalore	077-25814763
A008	Alford	New York	044-25874365
A011	Ravi Kumar	Bangalore	077-45625874
A012	Lucida	San Jose	044-52981425
A005	Anderson	Brisban	045-21447739

City is changed in 'Maximum deals Leased' section:



Total area owned/leased

Year: 2018

Leased: 272277.96 SQ-M

Owned: 486892.37 SQ-M

Max Area leased

CA Countries: 4929567.02 SQ-M

WS Countries: 328025.85 SQ-M

Maximums deals leased

City: Chilliwack

Agent: Alford

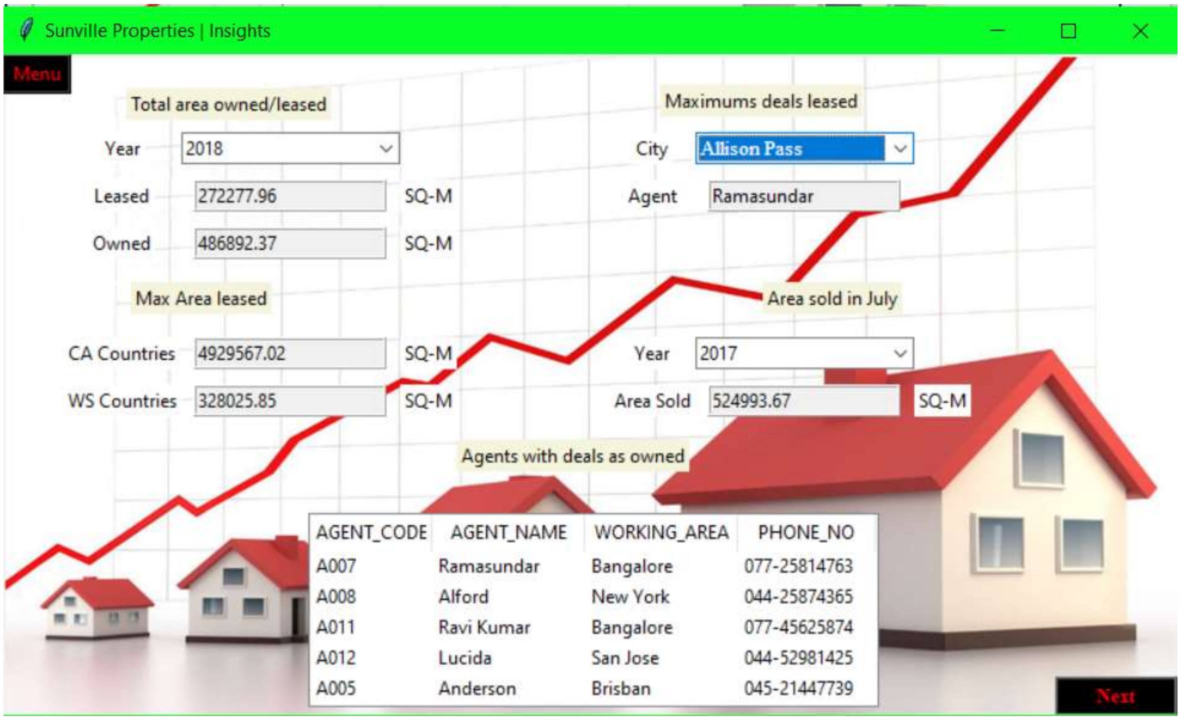
Area sold in July

Year: 2017

Area Sold: 524993.67 SQ-M

Agents with deals as owned

AGENT_CODE	AGENT_NAME	WORKING_AREA	PHONE_NO
A007	Ramasundar	Bangalore	077-25814763
A008	Alford	New York	044-25874365
A011	Ravi Kumar	Bangalore	077-45625874
A012	Lucida	San Jose	044-52981425
A005	Anderson	Brisban	045-21447739



Total area owned/leased

Year: 2018

Leased: 272277.96 SQ-M

Owned: 486892.37 SQ-M

Max Area leased

CA Countries: 4929567.02 SQ-M

WS Countries: 328025.85 SQ-M

Maximums deals leased

City: Allison Pass

Agent: Ramasundar

Area sold in July

Year: 2017

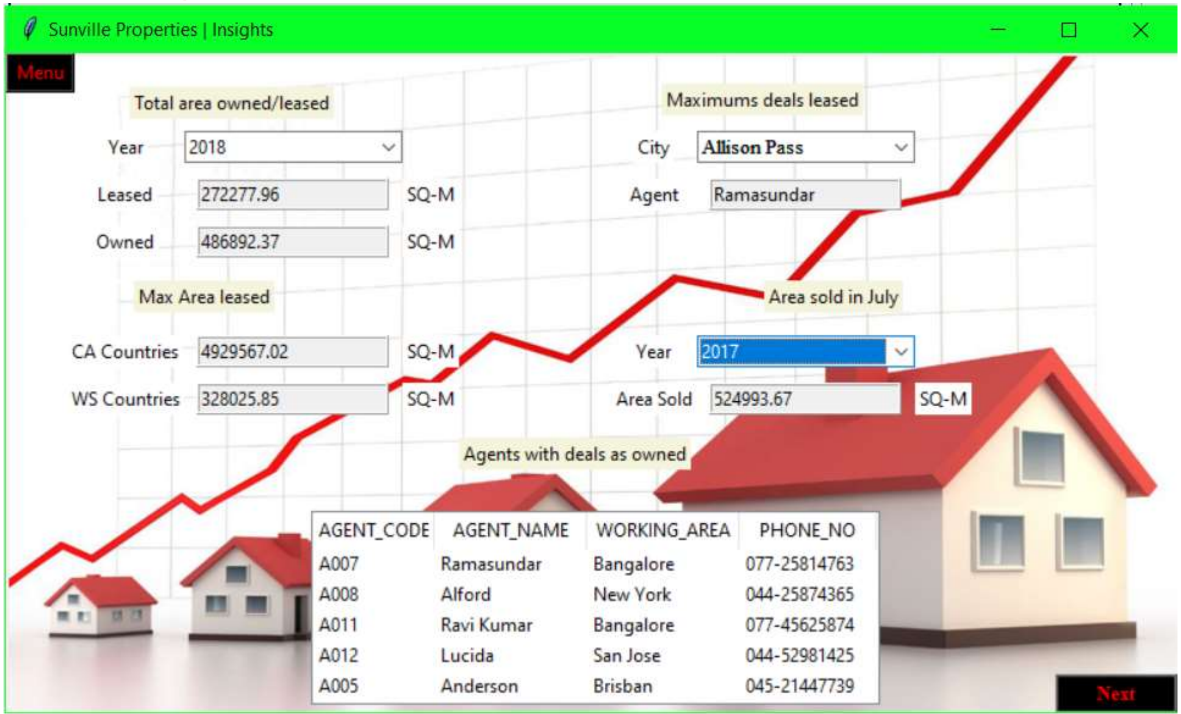
Area Sold: 524993.67 SQ-M

Agents with deals as owned

AGENT_CODE	AGENT_NAME	WORKING_AREA	PHONE_NO
A007	Ramasundar	Bangalore	077-25814763
A008	Alford	New York	044-25874365
A011	Ravi Kumar	Bangalore	077-45625874
A012	Lucida	San Jose	044-52981425
A005	Anderson	Brisban	045-21447739

Area sold in July:

Year is changed:



Total area owned/leased

Year: 2018

Leased: 272277.96 SQ-M

Owned: 486892.37 SQ-M

Maximums deals leased

City: Allison Pass

Agent: Ramasundar

Max Area leased

CA Countries: 4929567.02 SQ-M

WS Countries: 328025.85 SQ-M

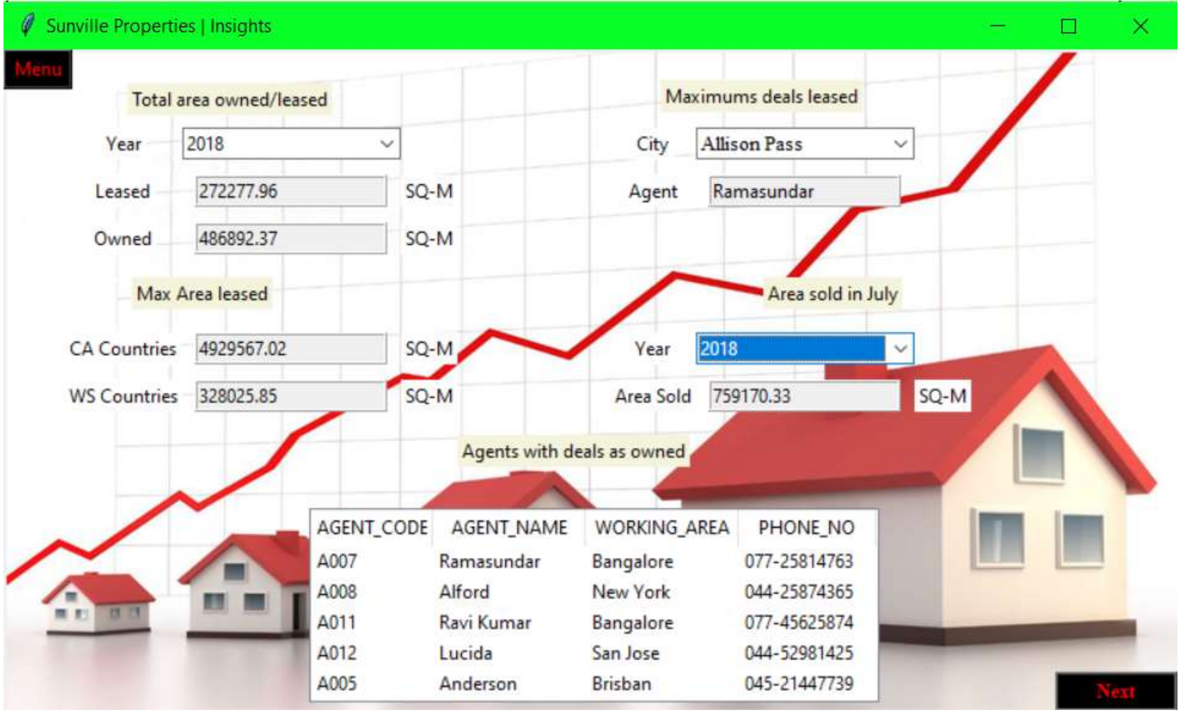
Area sold in July

Year: 2017

Area Sold: 524993.67 SQ-M

Agents with deals as owned

AGENT_CODE	AGENT_NAME	WORKING_AREA	PHONE_NO
A007	Ramasundar	Bangalore	077-25814763
A008	Alford	New York	044-25874365
A011	Ravi Kumar	Bangalore	077-45625874
A012	Lucida	San Jose	044-52981425
A005	Anderson	Brisban	045-21447739



Total area owned/leased

Year: 2018

Leased: 272277.96 SQ-M

Owned: 486892.37 SQ-M

Maximums deals leased

City: Allison Pass

Agent: Ramasundar

Max Area leased

CA Countries: 4929567.02 SQ-M

WS Countries: 328025.85 SQ-M

Area sold in July

Year: 2018

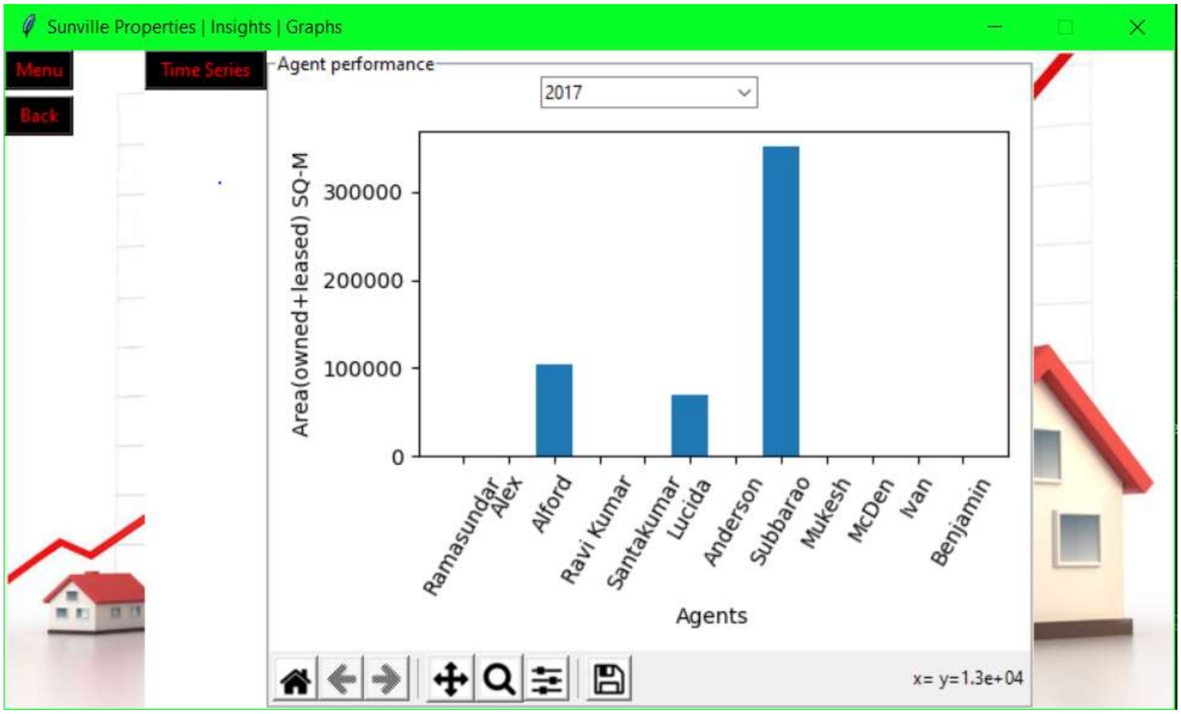
Area Sold: 759170.33 SQ-M

Agents with deals as owned

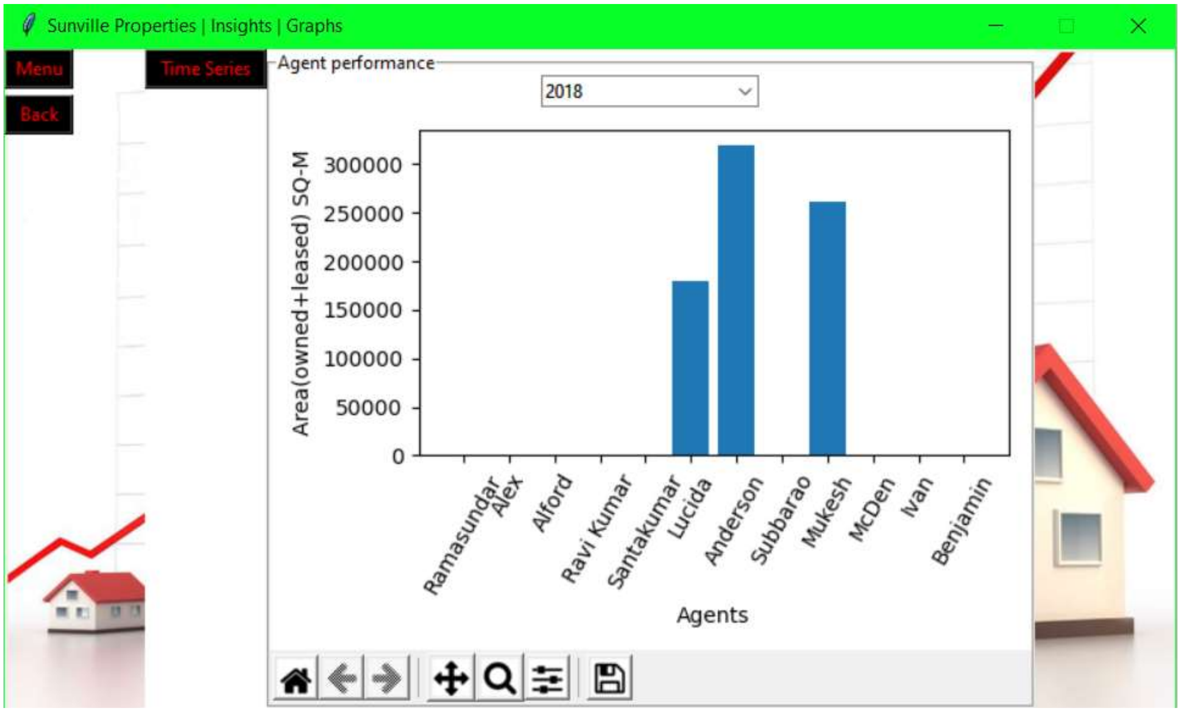
AGENT_CODE	AGENT_NAME	WORKING_AREA	PHONE_NO
A007	Ramasundar	Bangalore	077-25814763
A008	Alford	New York	044-25874365
A011	Ravi Kumar	Bangalore	077-45625874
A012	Lucida	San Jose	044-52981425
A005	Anderson	Brisban	045-21447739

Agent performance Next is clicked:

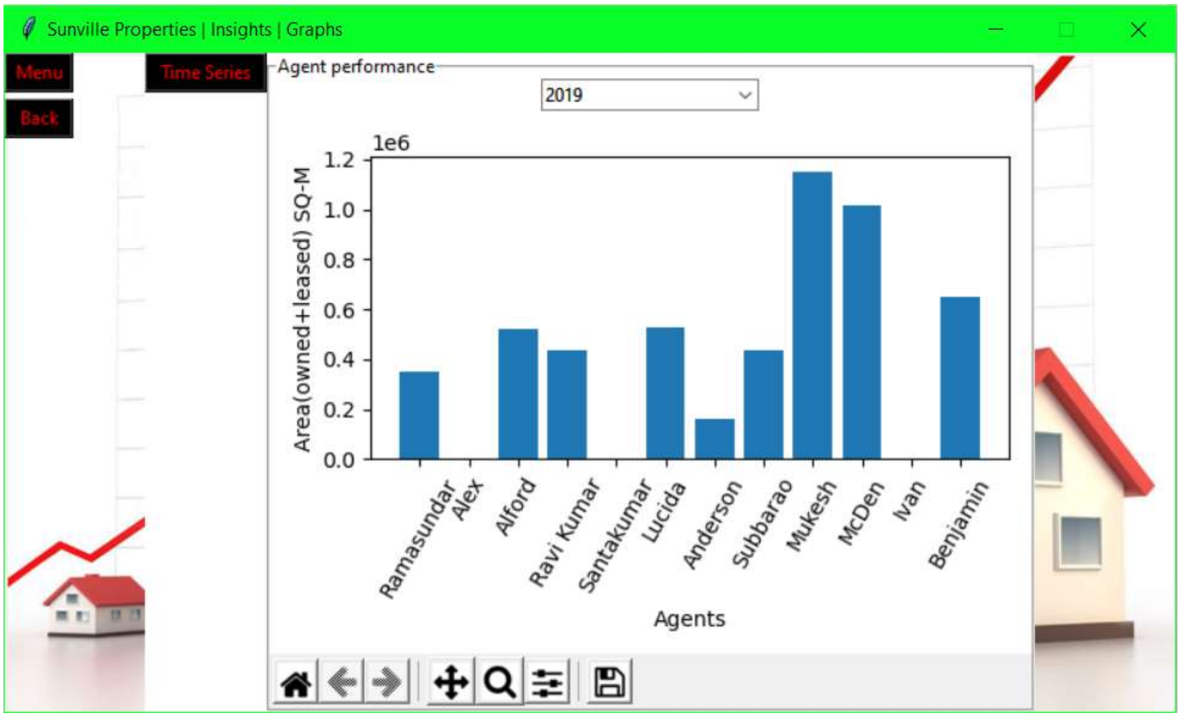
Year: 2017



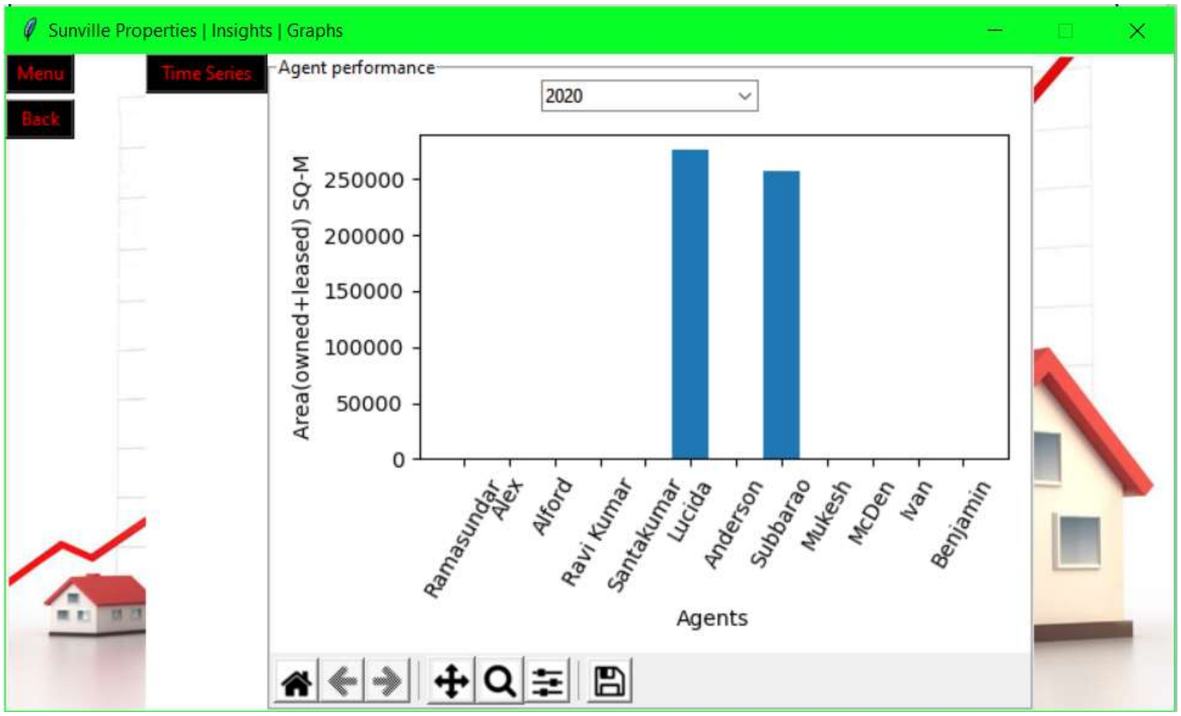
Year: 2018



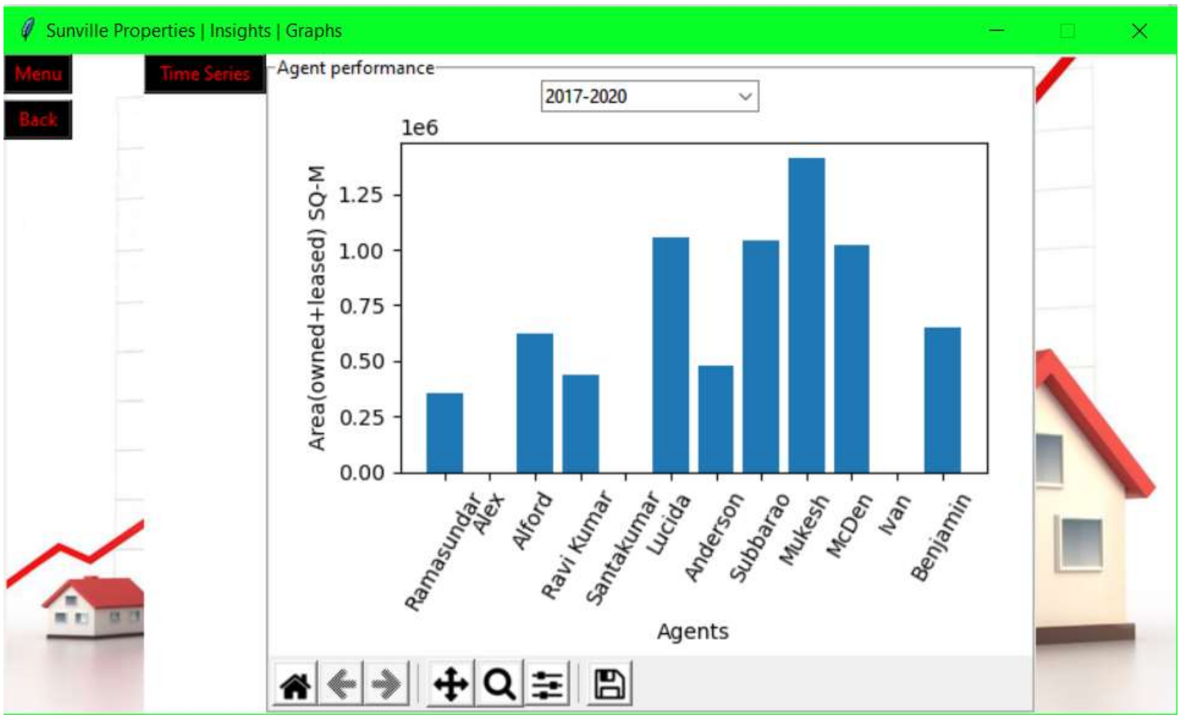
Year: 2019



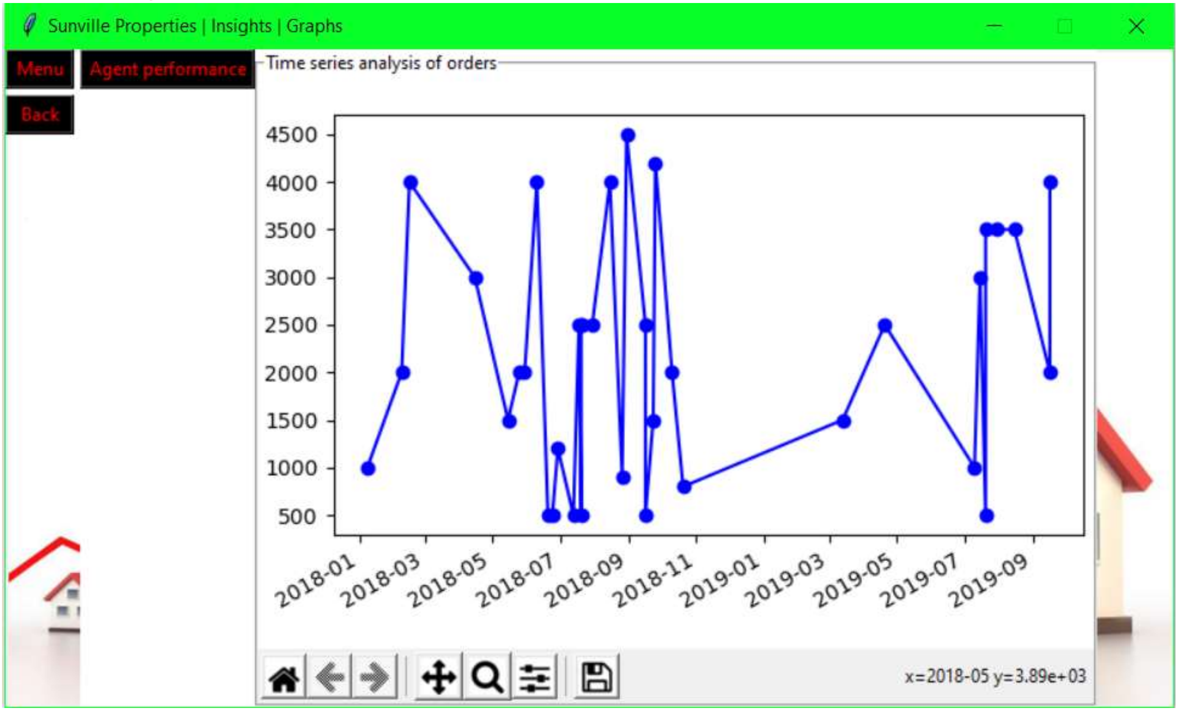
Year 2020:



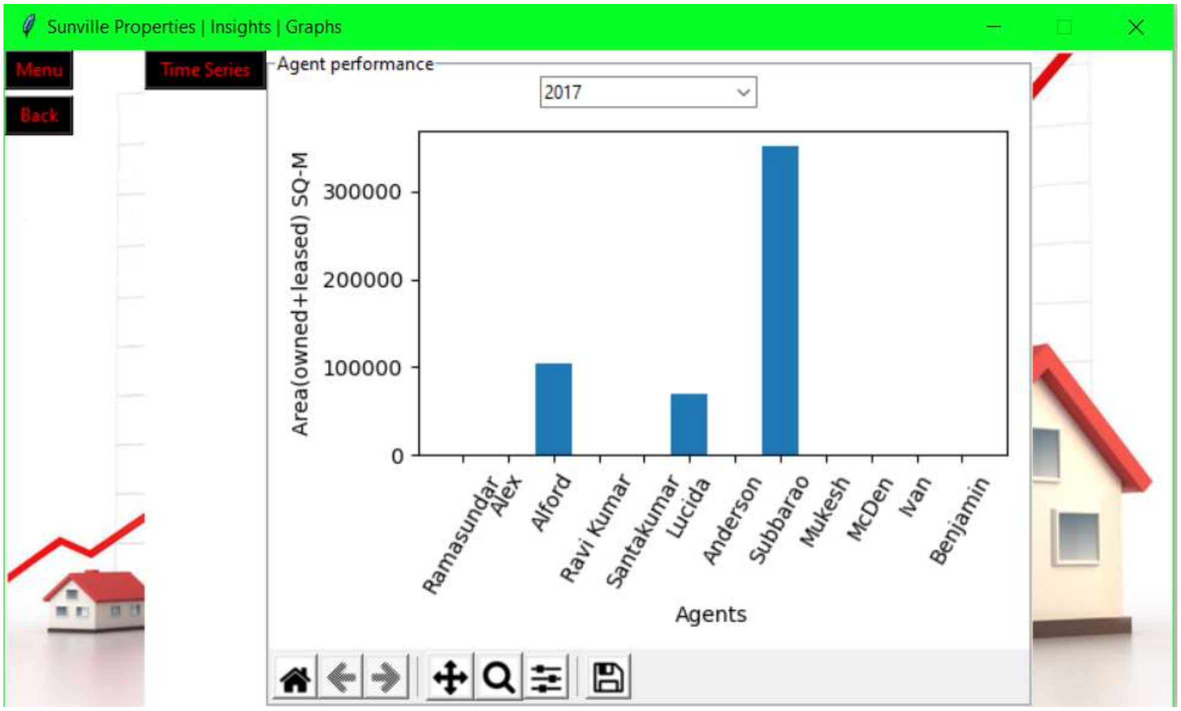
Year 2017-2020:



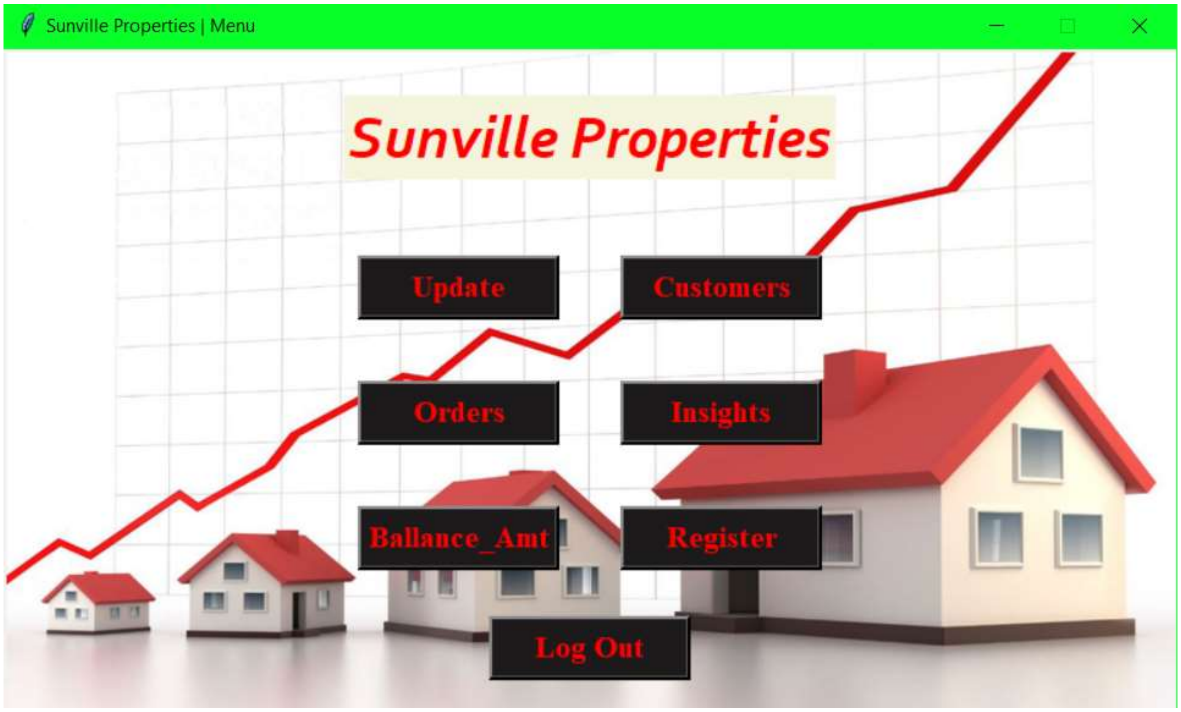
Time analysis button is clicked:

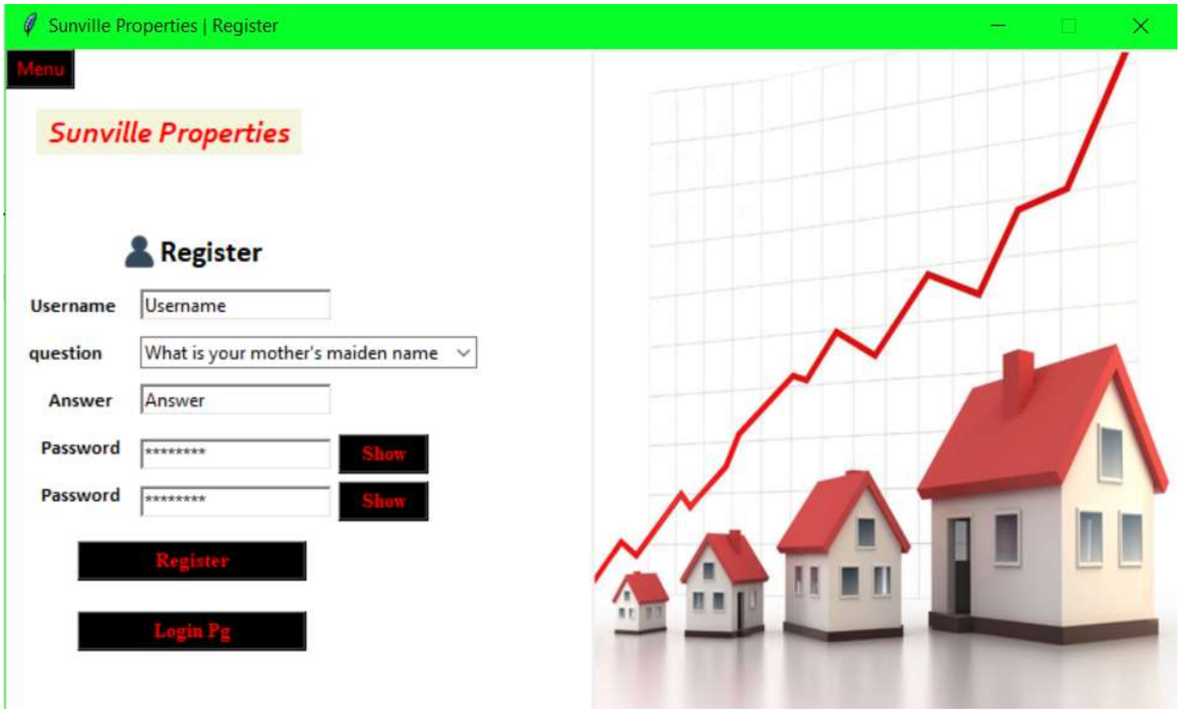


Agent performance Button is clicked again



Register Button is clicked:





Sunville Properties | Register

Register

Username: Username

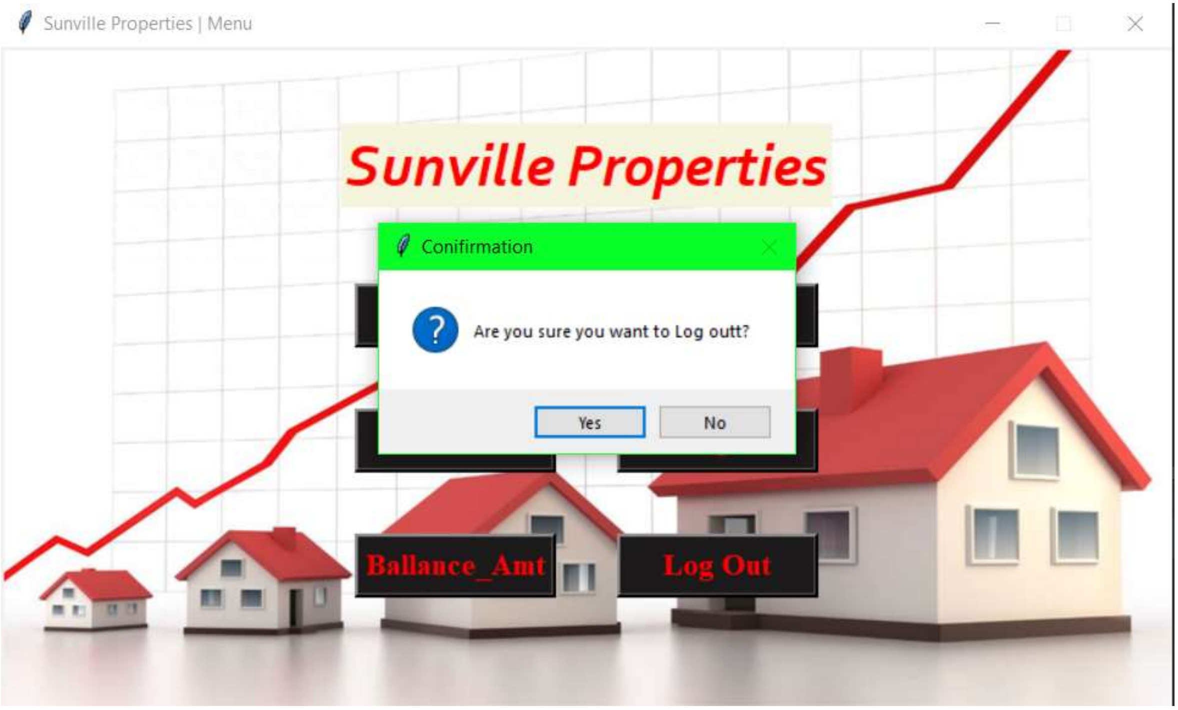
question: What is your mother's maiden name

Answer: Answer

Password: ********

Password: ********

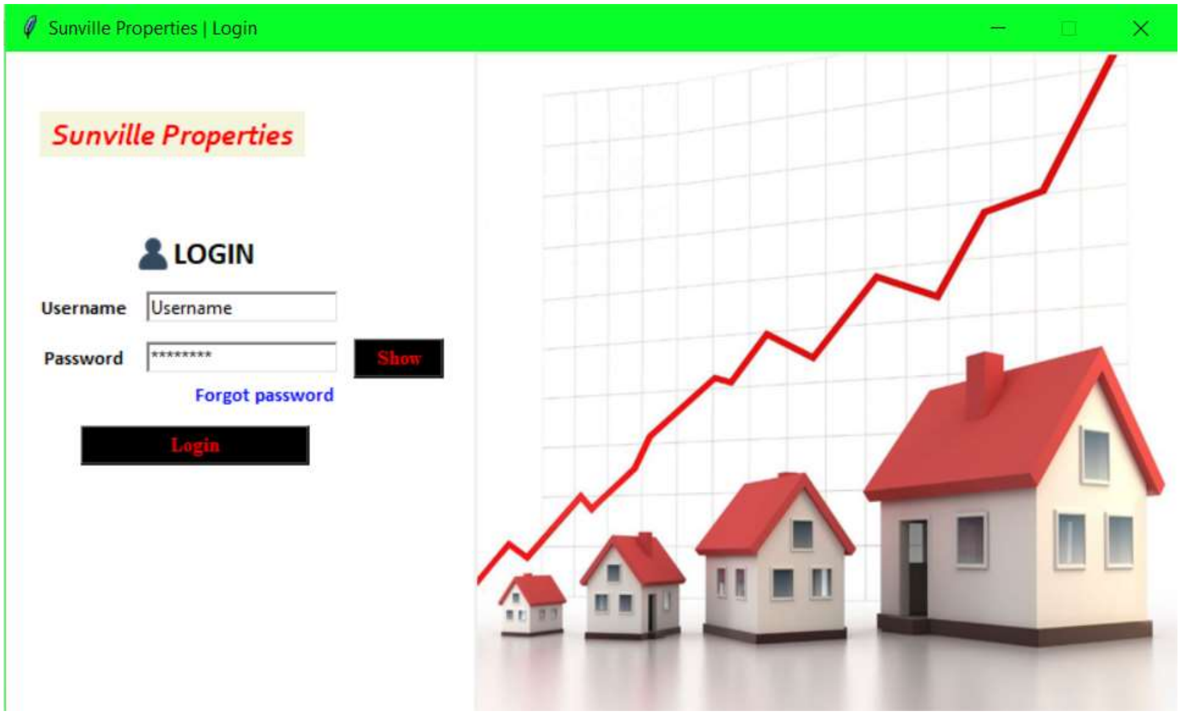
Log out is clicked on meu screen:



Sunville Properties | Menu

Conifirmation

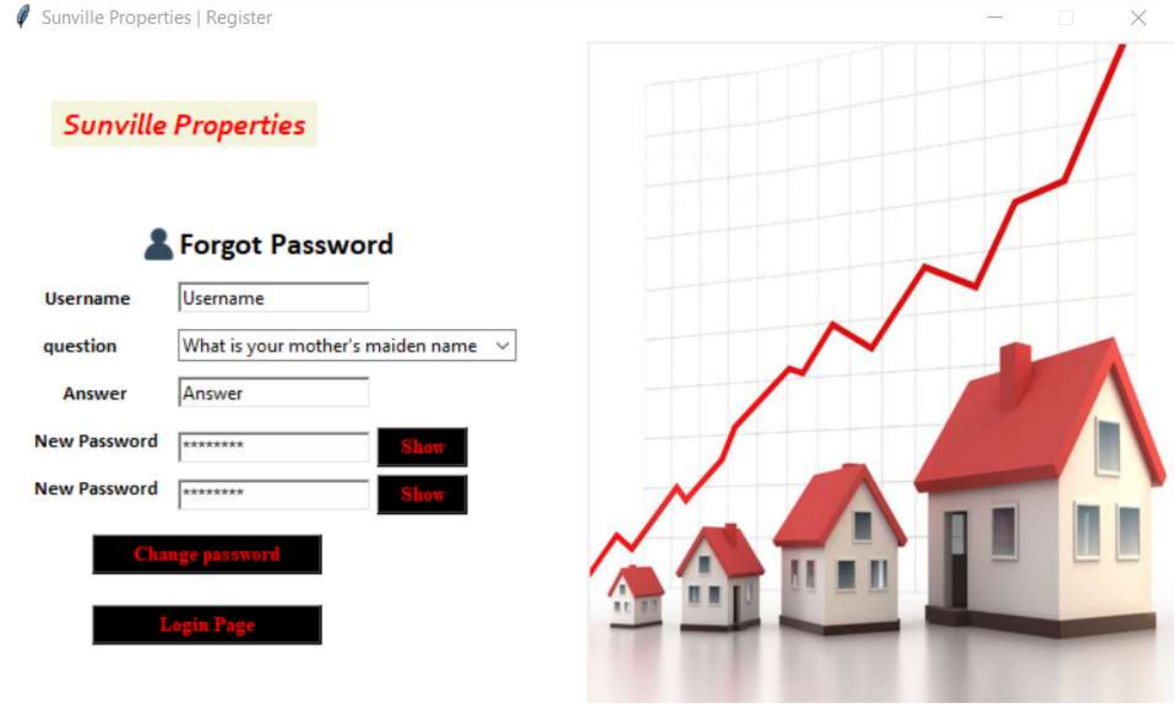
Are you sure you want to Log outt?



Username: Username

Password: ********

Forgot Password is clicked:



Username: Username

question: What is your mother's maiden name

Answer: Answer

New Password: ********

New Password: ********

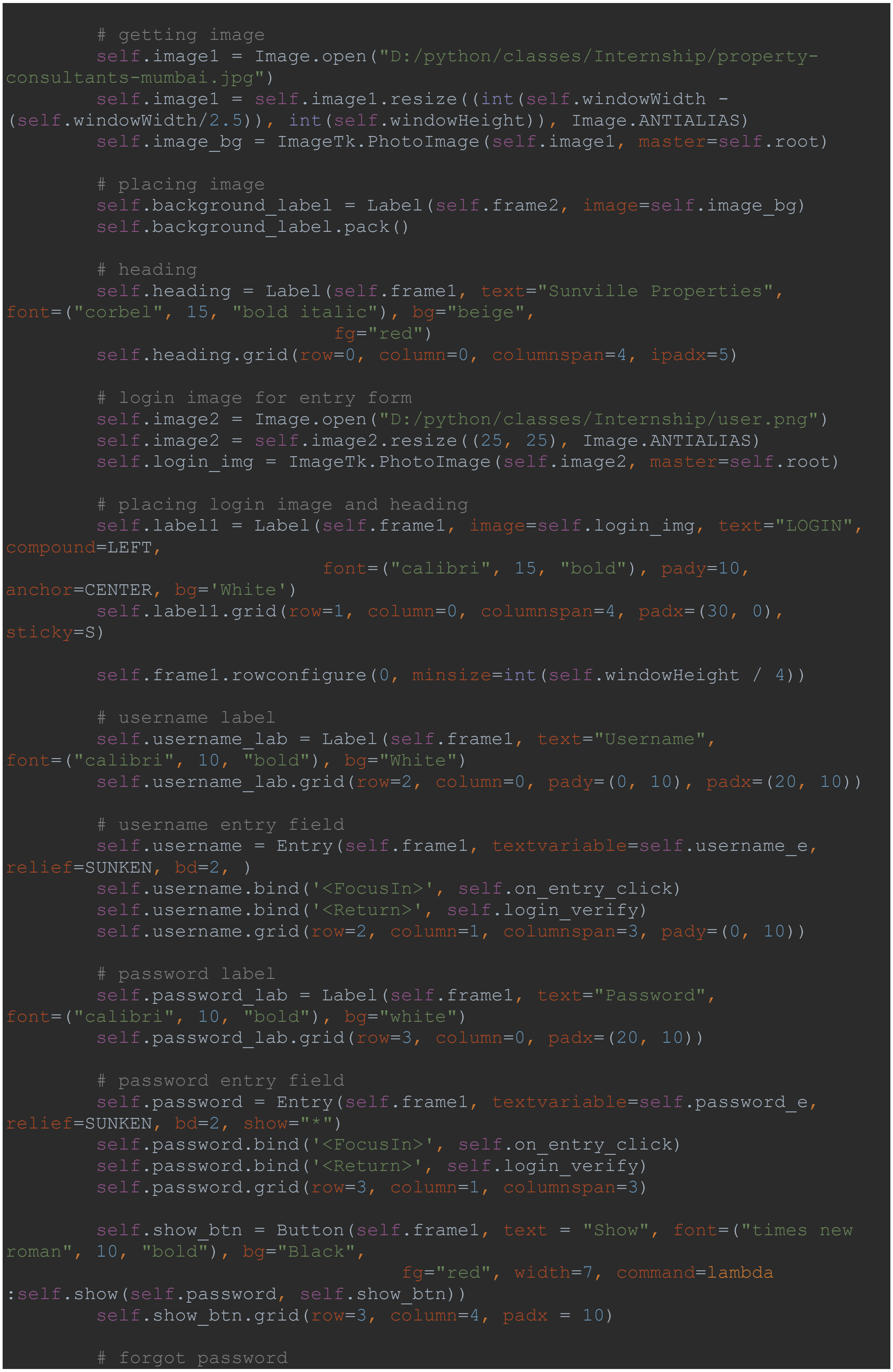
Section 6:

Final code:

https://github.com/gaurav2055/Sunville-Properties-python-internship

Login Module:

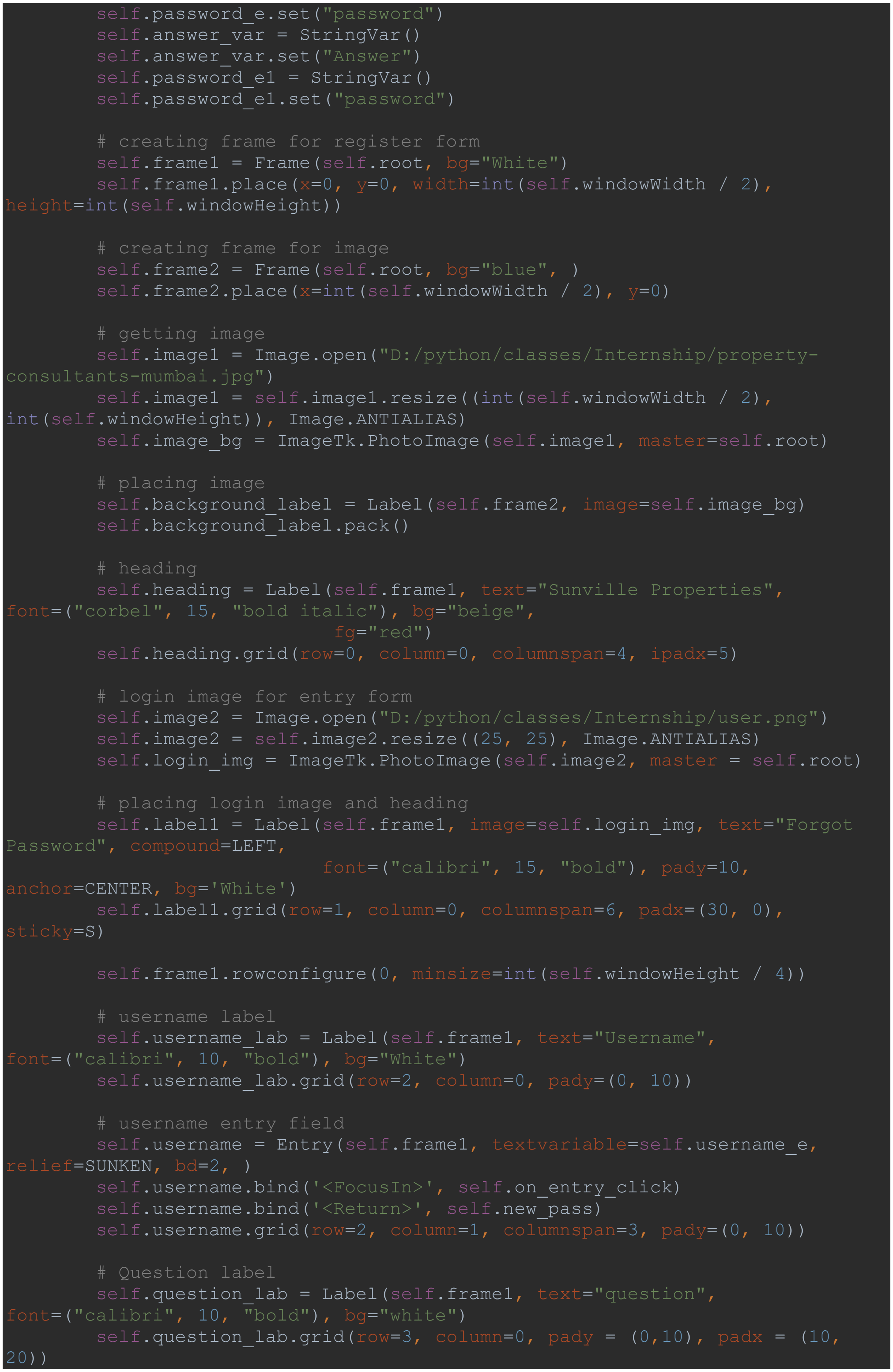
```
from tkinter import *
from PIL import ImageTk, Image
from tkinter import messagebox
import db.db
import menu
import forgot_password


class Login:
    def __init__(self):

        # creating tkinter window
        self.root = Tk()

        # setting window title
        self.root.title("Sunville Properties | Login")

        # determining size of window
        self.windowWidth = self.root.winfo_screenwidth() / 2
        self.windowHeight = self.root.winfo_screenheight() / 2

        # determining the the positon to set the window
        self.positionRight = int(self.root.winfo_screenwidth() / 2 -
self.windowWidth / 2)
        self.positionDown = int(self.root.winfo_screenheight() / 2 -
self.windowHeight / 2)

        # Positions the window in the center of the page.
        self.root.geometry(
            "{}x{}+{}+{}".format(int(self.windowWidth),
int(self.windowHeight), self.positionRight, self.positionDown))

        # disable resize of window
        self.root.resizable(False, False)

        self.FirstClick = True

    def login(self):

        # variable of entry fields
        self.username_e = StringVar()
        self.username_e.set("Username")
        self.password_e = StringVar()
        self.password_e.set("password")

        # creating frame for login form
        self.frame1 = Frame(self.root, bg="White")
        self.frame1.place(x=0, y=0, width=int(self.windowWidth / 2.5),
height=int(self.windowHeight))

        # creating frame for image
        self.frame2 = Frame(self.root, bg="blue", )
        self.frame2.place(x=int(self.windowWidth / 2.5), y=0)
```



```
        # getting image
        self.image1 = Image.open("D:/python/classes/Internship/property-
consultants-mumbai.jpg")
        self.image1 = self.image1.resize((int(self.windowWidth -
(self.windowWidth/2.5)), int(self.windowHeight)), Image.ANTIALIAS)
        self.image_bg = ImageTk.PhotoImage(self.image1, master=self.root)

        # placing image
        self.background_label = Label(self.frame2, image=self.image_bg)
        self.background_label.pack()

        # heading
        self.heading = Label(self.frame1, text="Sunville Properties",
font=("corbel", 15, "bold italic"), bg="beige",
                             fg="red")
        self.heading.grid(row=0, column=0, columnspan=4, ipadx=5)

        # login image for entry form
        self.image2 = Image.open("D:/python/classes/Internship/user.png")
        self.image2 = self.image2.resize((25, 25), Image.ANTIALIAS)
        self.login_img = ImageTk.PhotoImage(self.image2, master=self.root)

        # placing login image and heading
        self.label1 = Label(self.frame1, image=self.login_img, text="LOGIN",
compound=LEFT,
                            font=("calibri", 15, "bold"), pady=10,
anchor=CENTER, bg='White')
        self.label1.grid(row=1, column=0, columnspan=4, padx=(30, 0),
sticky=S)

        self.frame1.rowconfigure(0, minsize=int(self.windowHeight / 4))

        # username label
        self.username_lab = Label(self.frame1, text="Username",
font=("calibri", 10, "bold"), bg="White")
        self.username_lab.grid(row=2, column=0, pady=(0, 10), padx=(20, 10))

        # username entry field
        self.username = Entry(self.frame1, textvariable=self.username_e,
relief=SUNKEN, bd=2, )
        self.username.bind('<FocusIn>', self.on_entry_click)
        self.username.bind('<Return>', self.login_verify)
        self.username.grid(row=2, column=1, columnspan=3, pady=(0, 10))

        # password label
        self.password_lab = Label(self.frame1, text="Password",
font=("calibri", 10, "bold"), bg="white")
        self.password_lab.grid(row=3, column=0, padx=(20, 10))

        # password entry field
        self.password = Entry(self.frame1, textvariable=self.password_e,
relief=SUNKEN, bd=2, show="*")
        self.password.bind('<FocusIn>', self.on_entry_click)
        self.password.bind('<Return>', self.login_verify)
        self.password.grid(row=3, column=1, columnspan=3)

        self.show_btn = Button(self.frame1, text = "Show", font=("times new
roman", 10, "bold"), bg="Black",
                                   fg="red", width=7, command=lambda
:self.show(self.password, self.show_btn))
        self.show_btn.grid(row=3, column=4, padx = 10)

        # forgot password
        self.forgot_lab = Label(self.frame1, text = "Forgot password", font =
("Calibri", 10, "bold"), bg = "White", fg = "Blue")
        self.forgot_lab.grid(row=4, column = 3, sticky = N+E)
        self.forgot_lab.bind('<Button-1>', self.forgot)

        # login button
        self.login_button = Button(self.frame1, text="Login", font=("times
new roman", 10, "bold"), bg="Black",
                                   fg="red",
                                   width=20, command=self.login_verify)
        self.login_button.grid(row=5, column=0, columnspan=4, padx=(30, 0),
pady=10, sticky=N)

        self.root.mainloop()
    def show(self, widget, widget1, event=None):
        widget.config(show="")
        widget1.config(text = "Hide", command = lambda
:self.hide(widget,widget1))

    def hide(self, widget, widget1, event=None):
        widget.config(show="*")
        widget1.config(text = "Show", command = lambda
:self.show(widget,widget1))

    def on_entry_click(self, event):

        if self.FirstClick:
            self.FirstClick = False
            # delete all the text in entry fields
            self.username.delete(0, 'end')
            self.password.delete(0, 'end')

    def login_verify(self, event=None):

        # getting username and password entered
        self.username_info = self.username.get()
        self.password_info = self.password.get()

        # checking if all fields are full
        if self.username_info == "":
            messagebox.showerror("error", "username can not be blank")
            self.username_e.set("Username")
            self.password_e.set("Password")
            self.FirstClick = True
            self.username_lab.focus()

        elif self.password_info == "":
            messagebox.showerror("error", "Password can not be blank")
            self.password.focus()

        else:

            db.db.cursor.execute("SELECT * FROM `login` WHERE `Username` =
'%s' AND `Password`= md5('%s')" % (
            self.username_info, self.password_info))
            records = db.db.cursor.fetchall()

            if records:
                # destroy current window
                self.root.destroy()

                # open new window
                Menu = menu.Menu()
                Menu.menu()
            else:
                messagebox.showerror("Error", "Wrong Username or Password")
                # self.username_e.set("Username")
                self.password.delete(0, 'end')
                self.password.focus()

    def forgot(self, event=None):
        self.root.destroy()
        forgot = forgot_password.Forgot()
        forgot.forgot()


if __name__ == "__main__":
    x = Login()
    x.login()
```

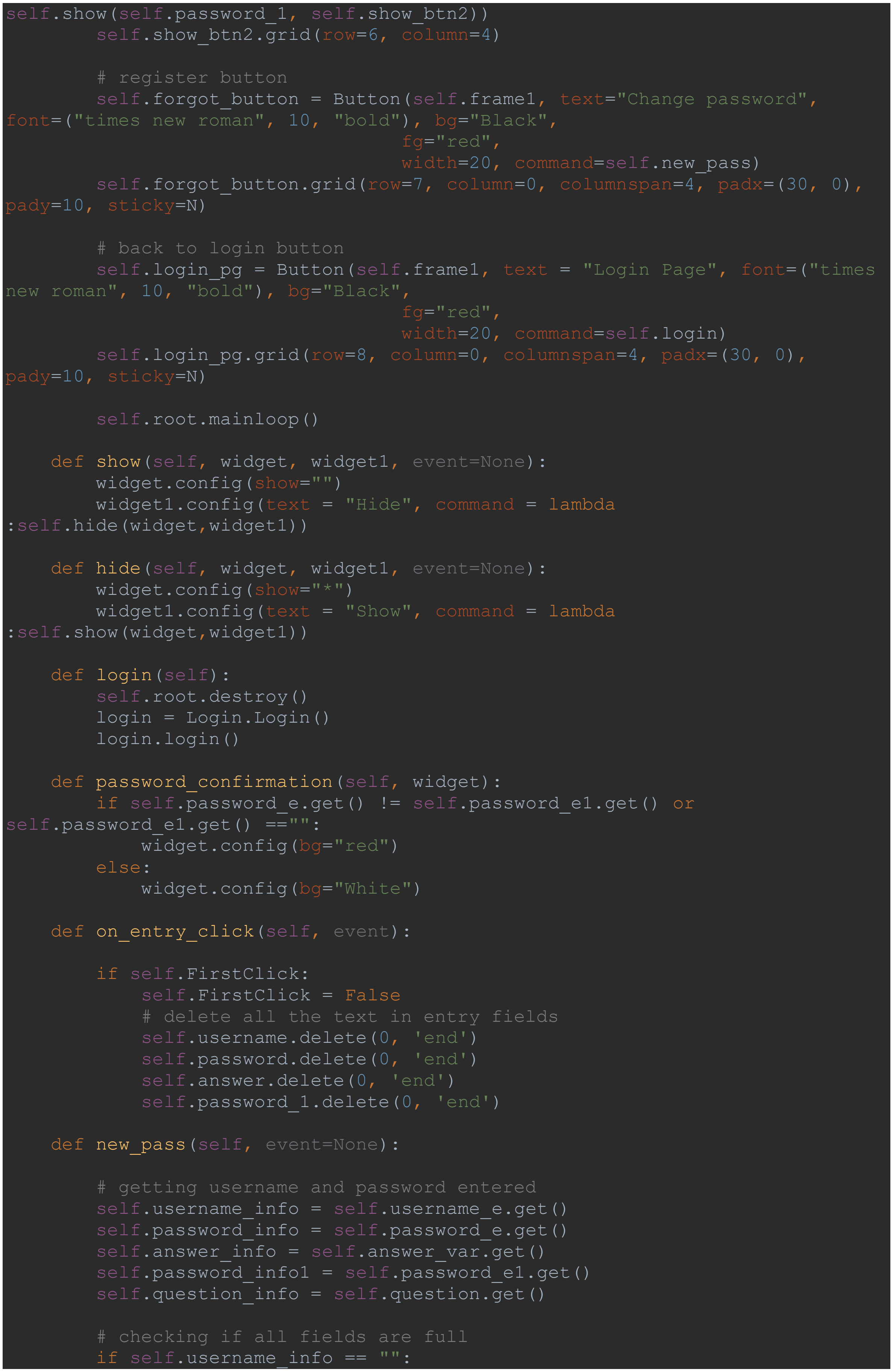
Forgot Password Module:

```
from tkinter import *
from PIL import ImageTk, Image
from tkinter import messagebox
from tkinter import ttk
import db.db
import Login


class Forgot:
    def __init__(self):

        # creating tkinter window
        self.root = Tk()

        # setting window title
        self.root.title("Sunville Properties | Register")

        # determining size of window
        self.windowWidth = self.root.winfo_screenwidth() / 2
        self.windowHeight = self.root.winfo_screenheight() / 2

        # determining the the positon to set the window
        self.positionRight = int(self.root.winfo_screenwidth() / 2 -
self.windowWidth / 2)
        self.positionDown = int(self.root.winfo_screenheight() / 2 -
self.windowHeight / 2)

        # Positions the window in the center of the page.
        self.root.geometry(
            "{}x{}+{}+{}".format(int(self.windowWidth),
int(self.windowHeight), self.positionRight, self.positionDown))

        # disable resize of window
        self.root.resizable(False, False)

        self.FirstClick = True

    def forgot(self):

        # variable of entry fields
        self.username_e = StringVar()
        self.username_e.set("Username")
        self.password_e = StringVar()
```



```
        self.password_e.set("password")
        self.answer_var = StringVar()
        self.answer_var.set("Answer")
        self.password_e1 = StringVar()
        self.password_e1.set("password")

        # creating frame for register form
        self.frame1 = Frame(self.root, bg="White")
        self.frame1.place(x=0, y=0, width=int(self.windowWidth / 2),
height=int(self.windowHeight))

        # creating frame for image
        self.frame2 = Frame(self.root, bg="blue", )
        self.frame2.place(x=int(self.windowWidth / 2), y=0)

        # getting image
        self.image1 = Image.open("D:/python/classes/Internship/property-
consultants-mumbai.jpg")
        self.image1 = self.image1.resize((int(self.windowWidth / 2),
int(self.windowHeight)), Image.ANTIALIAS)
        self.image_bg = ImageTk.PhotoImage(self.image1, master=self.root)

        # placing image
        self.background_label = Label(self.frame2, image=self.image_bg)
        self.background_label.pack()

        # heading
        self.heading = Label(self.frame1, text="Sunville Properties",
font=("corbel", 15, "bold italic"), bg="beige",
                             fg="red")
        self.heading.grid(row=0, column=0, columnspan=4, ipadx=5)

        # login image for entry form
        self.image2 = Image.open("D:/python/classes/Internship/user.png")
        self.image2 = self.image2.resize((25, 25), Image.ANTIALIAS)
        self.login_img = ImageTk.PhotoImage(self.image2, master = self.root)

        # placing login image and heading
        self.label1 = Label(self.frame1, image=self.login_img, text="Forgot
Password", compound=LEFT,
                            font=("calibri", 15, "bold"), pady=10,
anchor=CENTER, bg='White')
        self.label1.grid(row=1, column=0, columnspan=6, padx=(30, 0),
sticky=S)

        self.frame1.rowconfigure(0, minsize=int(self.windowHeight / 4))

        # username label
        self.username_lab = Label(self.frame1, text="Username",
font=("calibri", 10, "bold"), bg="White")
        self.username_lab.grid(row=2, column=0, pady=(0, 10))

        # username entry field
        self.username = Entry(self.frame1, textvariable=self.username_e,
relief=SUNKEN, bd=2, )
        self.username.bind('<FocusIn>', self.on_entry_click)
        self.username.bind('<Return>', self.new_pass)
        self.username.grid(row=2, column=1, columnspan=3, pady=(0, 10))

        # Question label
        self.question_lab = Label(self.frame1, text="question",
font=("calibri", 10, "bold"), bg="white")
        self.question_lab.grid(row=3, column=0, pady = (0,10), padx = (10,
20))
```

```
        # questions list
        questions = ["What is your mother's maiden name", "What is the name
of your first pet", "What is the name of "

"your first pet",
                     "What is your father's middle name"]

        # questions drop down
        self.question = ttk.Combobox(self.frame1, width = 33, values =
questions, state = "readonly")
        self.question.grid(row=3, column=3, columnspan=5, pady = (0, 10))
        self.question.current(0)

        # answer Label
        self.answer_lab = Label(self.frame1, text="Answer", font=("calibri",
10, "bold"), bg="white")
        self.answer_lab.grid(row = 4, column = 0, padx = (20, 10), pady = (0,
10))

        # answer entry field
        self.answer = Entry(self.frame1, textvariable = self.answer_var,
relief=SUNKEN, bd=2)
        self.answer.bind('<FocusIn>', self.on_entry_click)
        self.answer.bind('<Return>', self.new_pass)
        self.answer.grid(row=4, column=1, columnspan=3, pady = (0, 10))

        # password label
        self.password_lab = Label(self.frame1, text="New Password",
font=("calibri", 10, "bold"), bg="white")
        self.password_lab.grid(row=5, column=0, padx=(20, 10), pady=(0, 10))

        # password entry field
        self.password = Entry(self.frame1, textvariable=self.password_e,
relief=SUNKEN, bd=2, show="*")
        self.password.bind('<FocusIn>', self.on_entry_click)
        self.password.bind('<Return>', self.new_pass)
        self.password.grid(row=5, column=1, columnspan=3)

        # show password button1
        self.show_btn1 = Button(self.frame1, text = "Show", font=("times new
roman", 10, "bold"), bg="Black",
                                   fg="red", width=7, command=lambda
:self.show(self.password, self.show_btn1))
        self.show_btn1.grid(row=5, column=4)

        # password confirmation label
        self.password_lab1 = Label(self.frame1, text="New Password",
font=("calibri", 10, "bold"), bg="white")
        self.password_lab1.grid(row=6, column=0, padx=(20, 10), pady=(0, 10))

        # password confirmation entry field
        self.password_1 = Entry(self.frame1, textvariable=self.password_e1,
relief=SUNKEN, bd=2, show="*")
        self.password_1.bind('<FocusIn>', self.on_entry_click)
        self.password_1.bind('<FocusOut>', lambda e:
self.password_confirmation(self.password_1))
        self.password_1.bind('<Return>', self.new_pass)
        self.password_1.grid(row=6, column=1, columnspan=3)

        # show password button2
        self.show_btn2 = Button(self.frame1, text="Show", font=("times new
roman", 10, "bold"), bg="Black",
                                fg="red", width=7, command=lambda:
```



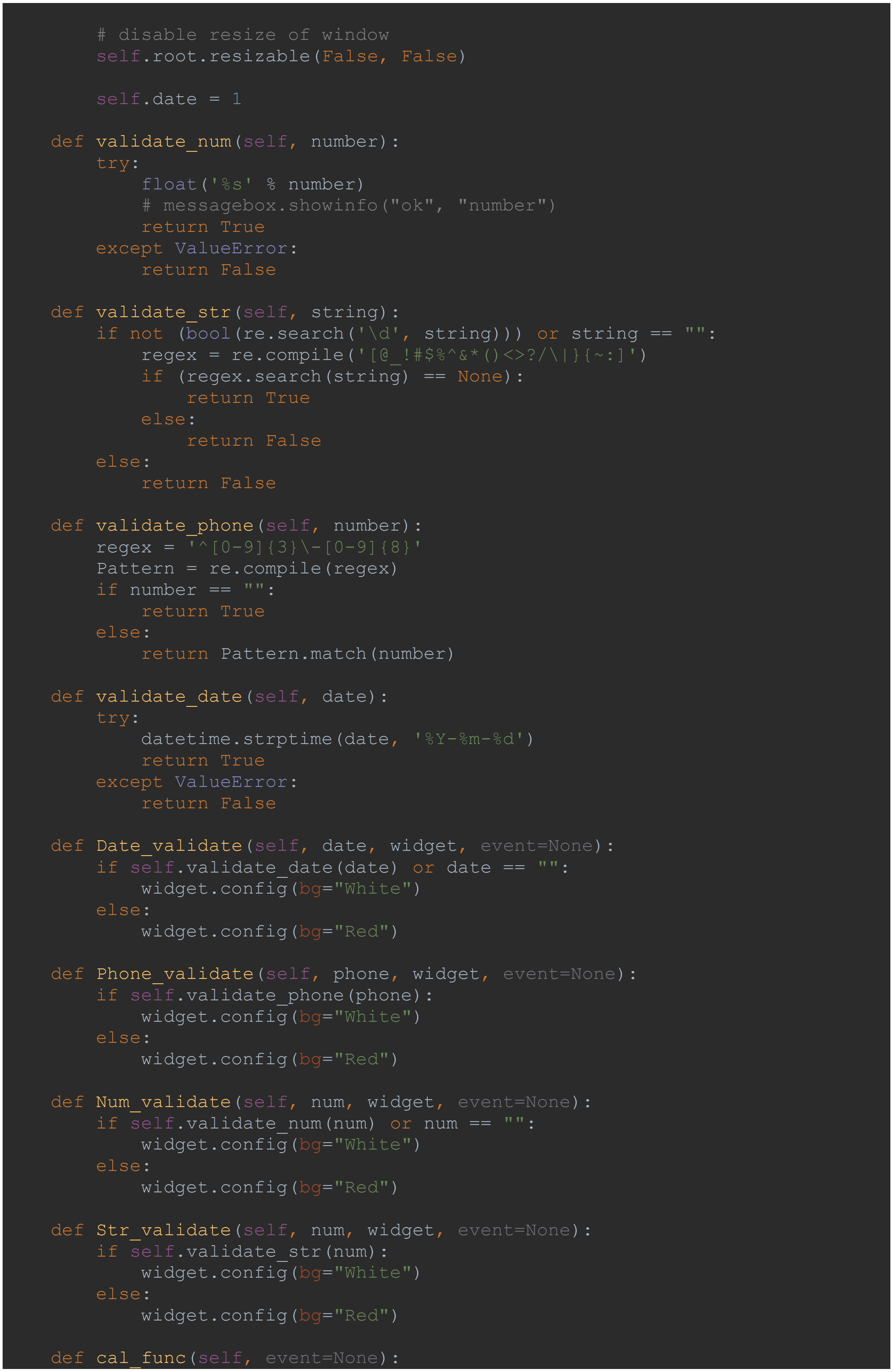
```
self.show(self.password_1, self.show_btn2))
        self.show_btn2.grid(row=6, column=4)

        # register button
        self.forgot_button = Button(self.frame1, text="Change password",
font=("times new roman", 10, "bold"), bg="Black",
                                   fg="red",
                                   width=20, command=self.new_pass)
        self.forgot_button.grid(row=7, column=0, columnspan=4, padx=(30, 0),
pady=10, sticky=N)

        # back to login button
        self.login_pg = Button(self.frame1, text = "Login Page", font=("times
new roman", 10, "bold"), bg="Black",
                                   fg="red",
                                   width=20, command=self.login)
        self.login_pg.grid(row=8, column=0, columnspan=4, padx=(30, 0),
pady=10, sticky=N)

        self.root.mainloop()

    def show(self, widget, widget1, event=None):
        widget.config(show="")
        widget1.config(text = "Hide", command = lambda
:self.hide(widget,widget1))

    def hide(self, widget, widget1, event=None):
        widget.config(show="*")
        widget1.config(text = "Show", command = lambda
:self.show(widget,widget1))

    def login(self):
        self.root.destroy()
        login = Login.Login()
        login.login()

    def password_confirmation(self, widget):
        if self.password_e.get() != self.password_e1.get() or
self.password_e1.get() =="":
            widget.config(bg="red")
        else:
            widget.config(bg="White")

    def on_entry_click(self, event):

        if self.FirstClick:
            self.FirstClick = False
            # delete all the text in entry fields
            self.username.delete(0, 'end')
            self.password.delete(0, 'end')
            self.answer.delete(0, 'end')
            self.password_1.delete(0, 'end')

    def new_pass(self, event=None):

        # getting username and password entered
        self.username_info = self.username_e.get()
        self.password_info = self.password_e.get()
        self.answer_info = self.answer_var.get()
        self.password_info1 = self.password_e1.get()
        self.question_info = self.question.get()

        # checking if all fields are full
        if self.username_info == "":
            messagebox.showerror("error", "username can not be blank")
            self.username_e.set("Username")
            self.password_e.set("Password")
            self.password_e1.set("Password")
            self.answer_var.set("Answer")
            self.FirstClick = True
            self.username_lab.focus()

        elif self.answer_info =="":
            messagebox.showerror("error", "Answer can not be blank")
            self.answer.focus()

        elif self.password_info == "":
            messagebox.showerror("error", "Password can not be blank")
            self.password.focus()

        elif self.password_info1 == "":
            messagebox.showerror("error", "Password can not be blank")
            self.password_1.focus()

        else:

            db.db.cursor.execute("SELECT * FROM `login` WHERE `Username` = '%s'" % self.username_info)
            records = db.db.cursor.fetchall()

            if records:
                if self.password_info == self.password_info1:
                    if records[0][2] == self.question_info:
                        if records[0][3] == self.answer_info:
                            query = """UPDATE `login` SET `Password`=md5("%s") WHERE `Username` = "%s" """
                            db.db.cursor.execute(query %(self.password_info, self.username_info))

                            db.db.con.commit()
                            messagebox.showinfo("Success", "Password reset successfully")

                            self.root.destroy()
                            login = Login.Login()
                            login.login()
                        else:
                            messagebox.showerror("Error", "Incorrect answer")
                    else:
                        messagebox.showerror("Error", "The question does not match")
                else:
                    messagebox.showerror("Error", "Passwords don't match")

            else:
                messagebox.showerror("Error", "Username does not exist")



if __name__ == "__main__":
    x = Forgot()
    x.forgot()
```

Menu Module:

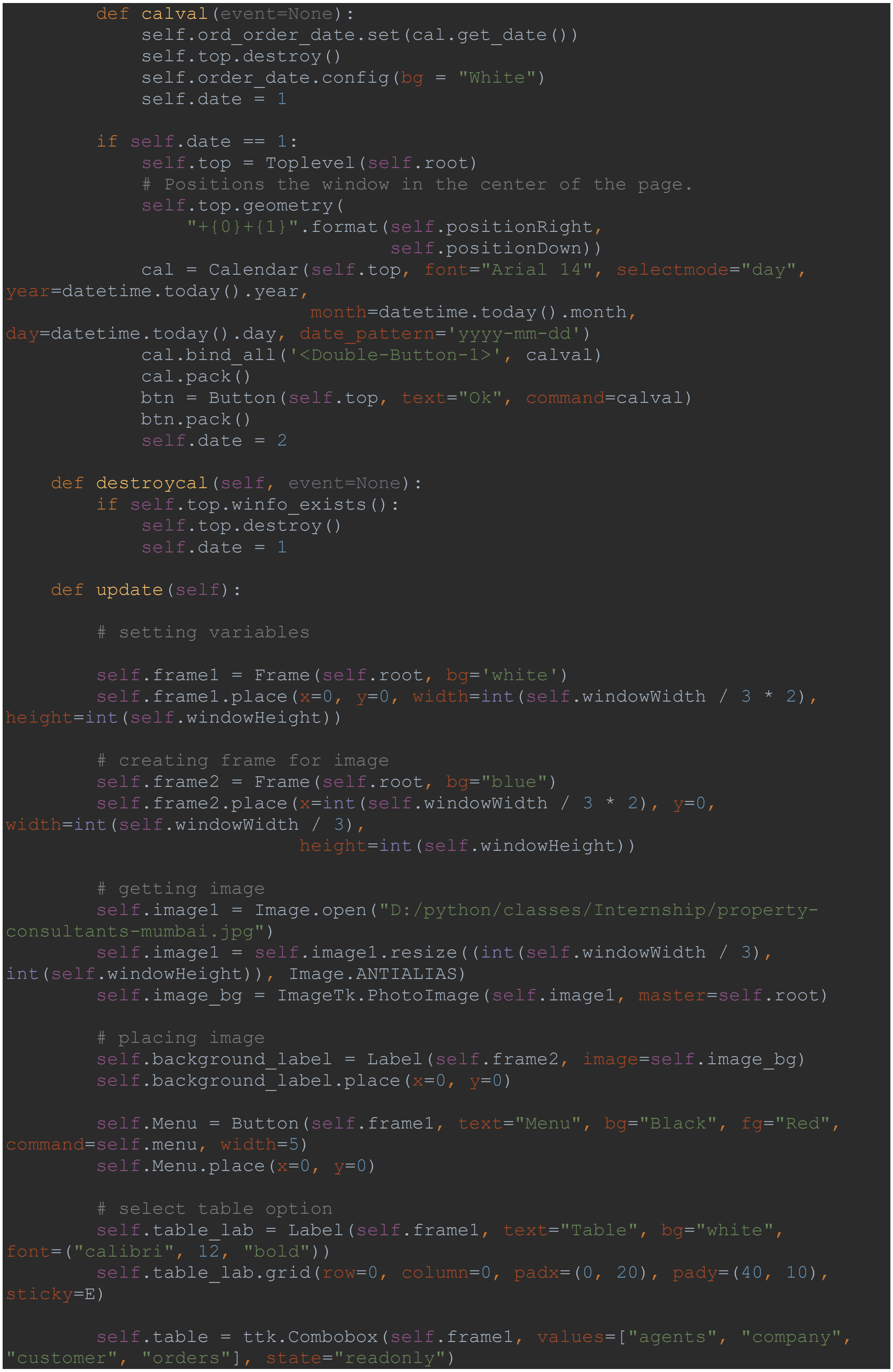
```
from tkinter import *
from PIL import ImageTk, Image
from tkinter import messagebox
import Login
import Register
import update
import orders_lookup
import balance_amount
import customers
import insights


class Menu:
    def __init__(self):
        # creating tkinter window
        self.root = Tk()

        # Setting title
        self.root.title("Sunville Properties | Menu")

        # determining size of the window
        self.windowWidth = self.root.winfo_screenwidth() / 2
        self.windowHeight = self.root.winfo_screenheight() / 2

        # determining the the positon to set the window
        self.positionRight = int(self.root.winfo_screenwidth() / 2 -
self.windowWidth / 2)
        self.positionDown = int(self.root.winfo_screenheight() / 2 -
self.windowHeight / 2)

        # Positions the window in the center of the page.
        self.root.geometry(
            "{}x{}+{}+{}".format(int(self.windowWidth),
int(self.windowHeight), self.positionRight, self.positionDown))

        # disable resize of window
        self.root.resizable(False, False)

    def menu(self):

        # getting image
        self.image1 = Image.open("D:/python/classes/Internship/property-
consultants-mumbai.jpg")
        self.image1 = self.image1.resize((int(self.windowWidth),
int(self.windowHeight)), Image.ANTIALIAS)
        self.image_bg = ImageTk.PhotoImage(self.image1, master = self.root)

        # placing image
        self.background_label = Label(self.root, image=self.image_bg)
        self.background_label.place(x=0, y=0)

        self.head_lab = Label(self.root, text = "Sunville Properties", font =
("corbel", 30,"bold italic"), bg = "beige", fg = "red")
        self.head_lab.grid(row = 0, column = 0, columnspan = 2, padx = 100,
pady = (30,30))

        self.root.rowconfigure(0, minsize=int(self.windowHeight / 4))
        self.root.columnconfigure(0, minsize = (self.windowWidth/ 2))
        self.root.columnconfigure(1, minsize=(self.windowWidth / 2))

        self.update_btn = Button(self.root, text="Update", font=("times new
roman", 15, "bold"), bg="#1C1B1B", fg="red",
                                 width=10, command=self.update,
relief=RAISED, bd=3)
        self.update_btn.grid(row=1, column=0, pady = 20, sticky = E, padx =
20)

        self.orders_btn = Button(self.root, text = "Orders", font=("times new
roman", 15, "bold"), bg="#1C1B1B", fg="red",
                                 width=10, command=self.orders,
relief=RAISED, bd=3)
        self.orders_btn.grid(row=2, column=0, pady = 20, sticky = E, padx =
20)

        self.balance_amt_btn = Button(self.root, text = "Ballance_Amt",
font=("times new roman", 15, "bold"), bg="#1C1B1B", fg="red",
                                 width=10, command=self.balamce_amt,
relief=RAISED, bd=3)
        self.balance_amt_btn.grid(row=3, column=0, pady = 20, sticky = E,
padx = 20)

        self.customers_btn = Button(self.root, text = "Customers",
font=("times new roman", 15, "bold"), bg="#1C1B1B", fg="red",
                                 width=10, command=self.customers,
relief=RAISED, bd=3)
        self.customers_btn.grid(row=1, column=1, pady = 20, sticky = W, padx
= 20)

        self.insights_btn = Button(self.root, text = "Insights", font=("times
new roman", 15, "bold"), bg="#1C1B1B", fg="red",
                                 width=10, command=self.insights,
relief=RAISED, bd=3)
        self.insights_btn.grid(row = 2, column = 1, pady = 20, sticky = W,
padx = 20)

        self.register_btn = Button(self.root, text = "Register", font=("times
new roman", 15, "bold"), bg="#1C1B1B", fg="red",
                                 width=10, command=self.register,
relief=RAISED, bd=3)
        self.register_btn.grid(row = 3, column = 1, pady = 20, sticky = W,
padx = 20)

        self.log_out_btn = Button(self.root, text = "Log Out", font=("times
new roman", 15, "bold"), bg="#1C1B1B", fg="red",
                                 width=10, command=self.log_out,
relief=RAISED, bd=3)
        self.log_out_btn.grid(row = 4, column = 0, pady = 10, columnspan=2)
        self.root.mainloop()

    def update(self):
        self.root.destroy()
        Update = update.Update()
        Update.update()
    def orders(self):
        self.root.destroy()
        orders = orders_lookup.orders_lookup()
        orders.orders()
    def balamce_amt(self):
        self.root.destroy()
        bal = balance_amount.ballance_amt()
        bal.balance()

    def customers(self):
        self.root.destroy()
        cus = customers.Customers()
        cus.customers()

    def insights(self):
        self.root.destroy()
        ins = insights.Insights()
        ins.insights()

    def register(self):
        self.root.destroy()
        reg = Register.Register()
        reg.register()

    def log_out(self):
        response = messagebox.askquestion("Conifirmation","Are you sure you
want to Log outt?")
        if response:
            self.root.destroy()
            log = Login.Login()
            log.login()

if __name__ == "__main__":
    x = Menu()
    x.menu()
```

Update Module:

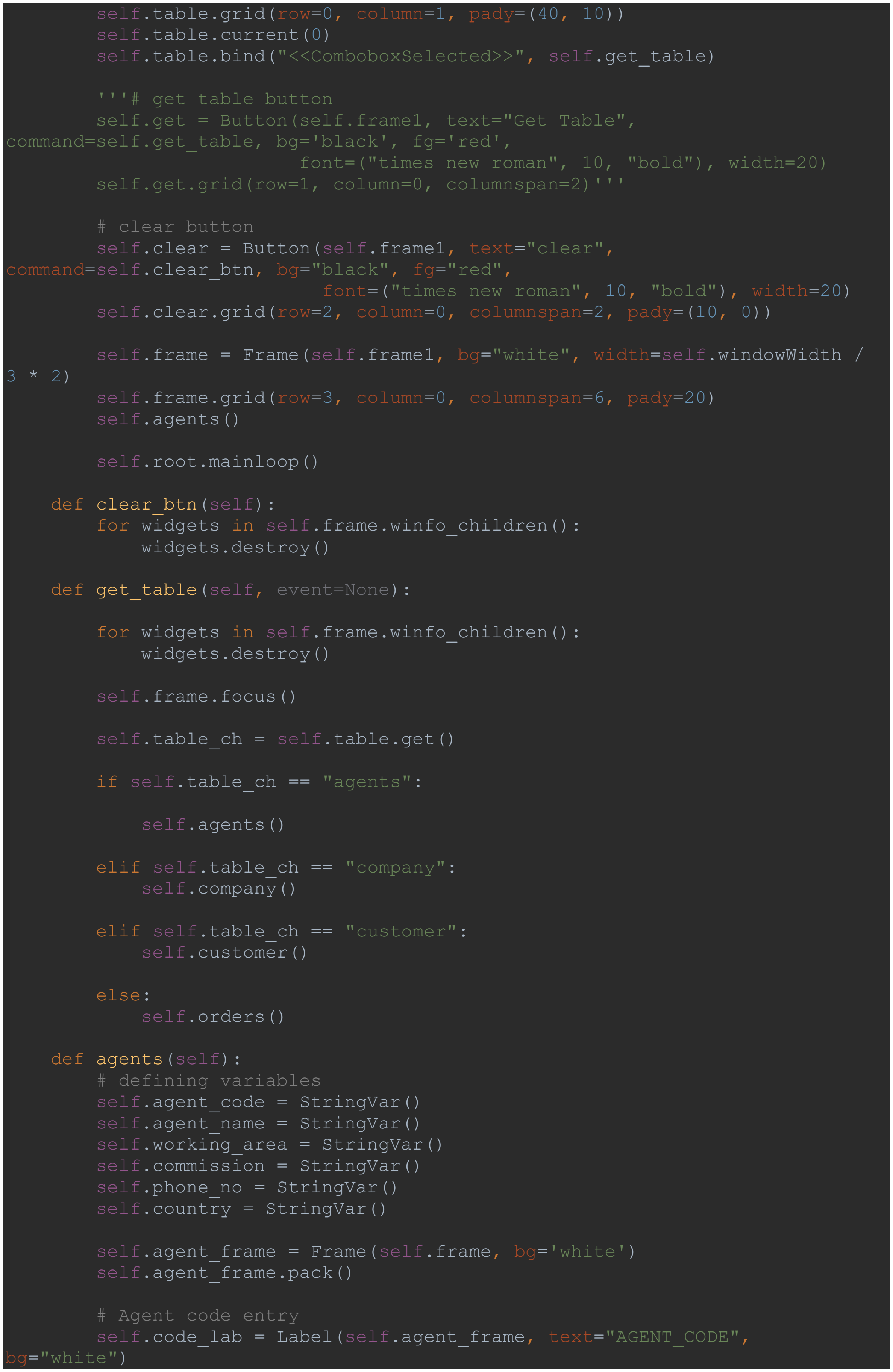
```
from tkinter import *
from PIL import ImageTk, Image
from tkinter import messagebox
from tkinter import ttk
from tkcalendar import *
from datetime import *
import re
import db.db
import menu


class Update:
    def __init__(self):

        # creating tkinter window
        self.root = Tk()
        self.top = Toplevel(self.root)
        self.top.destroy()

        self.root.config(bg="white")

        # Setting title
        self.root.title("Sunville Properties | Update")

        # determining size of window
        self.windowWidth = self.root.winfo_screenwidth() / 2
        self.windowHeight = self.root.winfo_screenheight() / 2

        # detriming the postion to set the window
        self.positionRight = int(self.root.winfo_screenwidth() / 2 -
self.windowWidth / 2)
        self.positionDown = int(self.root.winfo_screenheight() / 2 -
self.windowHeight / 2)

        # Positions the window in the center of the page.
        self.root.geometry("{0}x{1}+{2}+{3}".format(int(self.windowWidth),
int(self.windowHeight), self.positionRight,
                                                    self.positionDown))
```



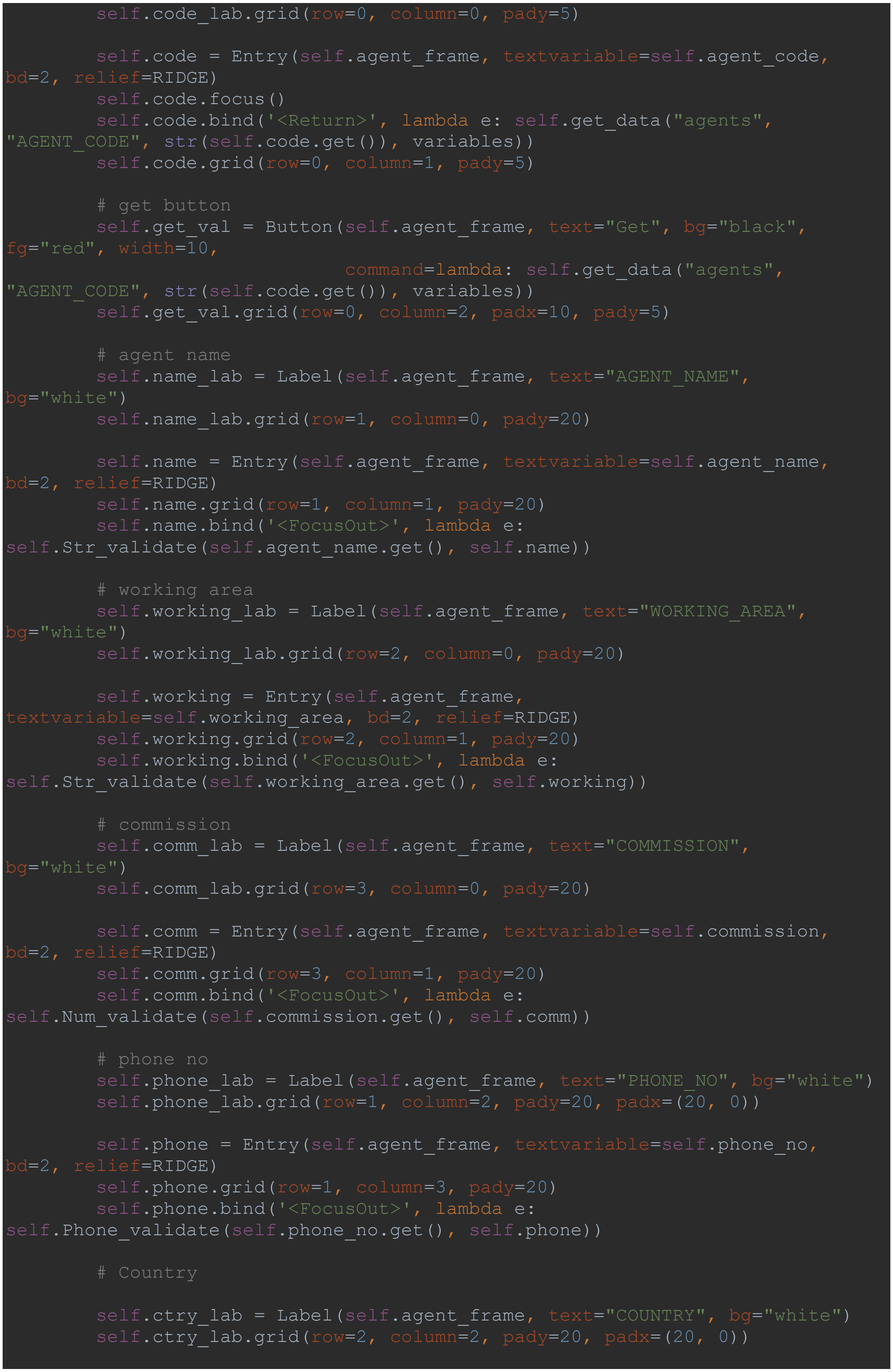
```python
        # disable resize of window
        self.root.resizable(False, False)

        self.date = 1

    def validate_num(self, number):
        try:
            float('%s' % number)
            # messagebox.showinfo("ok", "number")
            return True
        except ValueError:
            return False

    def validate_str(self, string):
        if not (bool(re.search('\d', string))) or string == "":
            regex = re.compile('[@_!#$%^&*()<>?/\|}{~:]')
            if (regex.search(string) == None):
                return True
            else:
                return False
        else:
            return False

    def validate_phone(self, number):
        regex = '^[0-9]{3}\-[0-9]{8}'
        Pattern = re.compile(regex)
        if number == "":
            return True
        else:
            return Pattern.match(number)

    def validate_date(self, date):
        try:
            datetime.strptime(date, '%Y-%m-%d')
            return True
        except ValueError:
            return False

    def Date_validate(self, date, widget, event=None):
        if self.validate_date(date) or date == "":
            widget.config(bg="White")
        else:
            widget.config(bg="Red")

    def Phone_validate(self, phone, widget, event=None):
        if self.validate_phone(phone):
            widget.config(bg="White")
        else:
            widget.config(bg="Red")

    def Num_validate(self, num, widget, event=None):
        if self.validate_num(num) or num == "":
            widget.config(bg="White")
        else:
            widget.config(bg="Red")

    def Str_validate(self, num, widget, event=None):
        if self.validate_str(num):
            widget.config(bg="White")
        else:
            widget.config(bg="Red")

    def cal_func(self, event=None):
```



```
        def calval(event=None):
            self.ord_order_date.set(cal.get_date())
            self.top.destroy()
            self.order_date.config(bg = "White")
            self.date = 1

        if self.date == 1:
            self.top = Toplevel(self.root)
            # Positions the window in the center of the page.
            self.top.geometry(
                "+{0}+{1}".format(self.positionRight,
                                  self.positionDown))
            cal = Calendar(self.top, font="Arial 14", selectmode="day",
year=datetime.today().year,
                           month=datetime.today().month,
day=datetime.today().day, date_pattern='yyyy-mm-dd')
            cal.bind_all('<Double-Button-1>', calval)
            cal.pack()
            btn = Button(self.top, text="Ok", command=calval)
            btn.pack()
            self.date = 2

    def destroycal(self, event=None):
        if self.top.winfo_exists():
            self.top.destroy()
            self.date = 1

    def update(self):

        # setting variables

        self.frame1 = Frame(self.root, bg='white')
        self.frame1.place(x=0, y=0, width=int(self.windowWidth / 3 * 2),
height=int(self.windowHeight))

        # creating frame for image
        self.frame2 = Frame(self.root, bg="blue")
        self.frame2.place(x=int(self.windowWidth / 3 * 2), y=0,
width=int(self.windowWidth / 3),
                          height=int(self.windowHeight))

        # getting image
        self.image1 = Image.open("D:/python/classes/Internship/property-
consultants-mumbai.jpg")
        self.image1 = self.image1.resize((int(self.windowWidth / 3),
int(self.windowHeight)), Image.ANTIALIAS)
        self.image_bg = ImageTk.PhotoImage(self.image1, master=self.root)

        # placing image
        self.background_label = Label(self.frame2, image=self.image_bg)
        self.background_label.place(x=0, y=0)

        self.Menu = Button(self.frame1, text="Menu", bg="Black", fg="Red",
command=self.menu, width=5)
        self.Menu.place(x=0, y=0)

        # select table option
        self.table_lab = Label(self.frame1, text="Table", bg="white",
font=("calibri", 12, "bold"))
        self.table_lab.grid(row=0, column=0, padx=(0, 20), pady=(40, 10),
sticky=E)

        self.table = ttk.Combobox(self.frame1, values=["agents", "company",
"customer", "orders"], state="readonly")
```

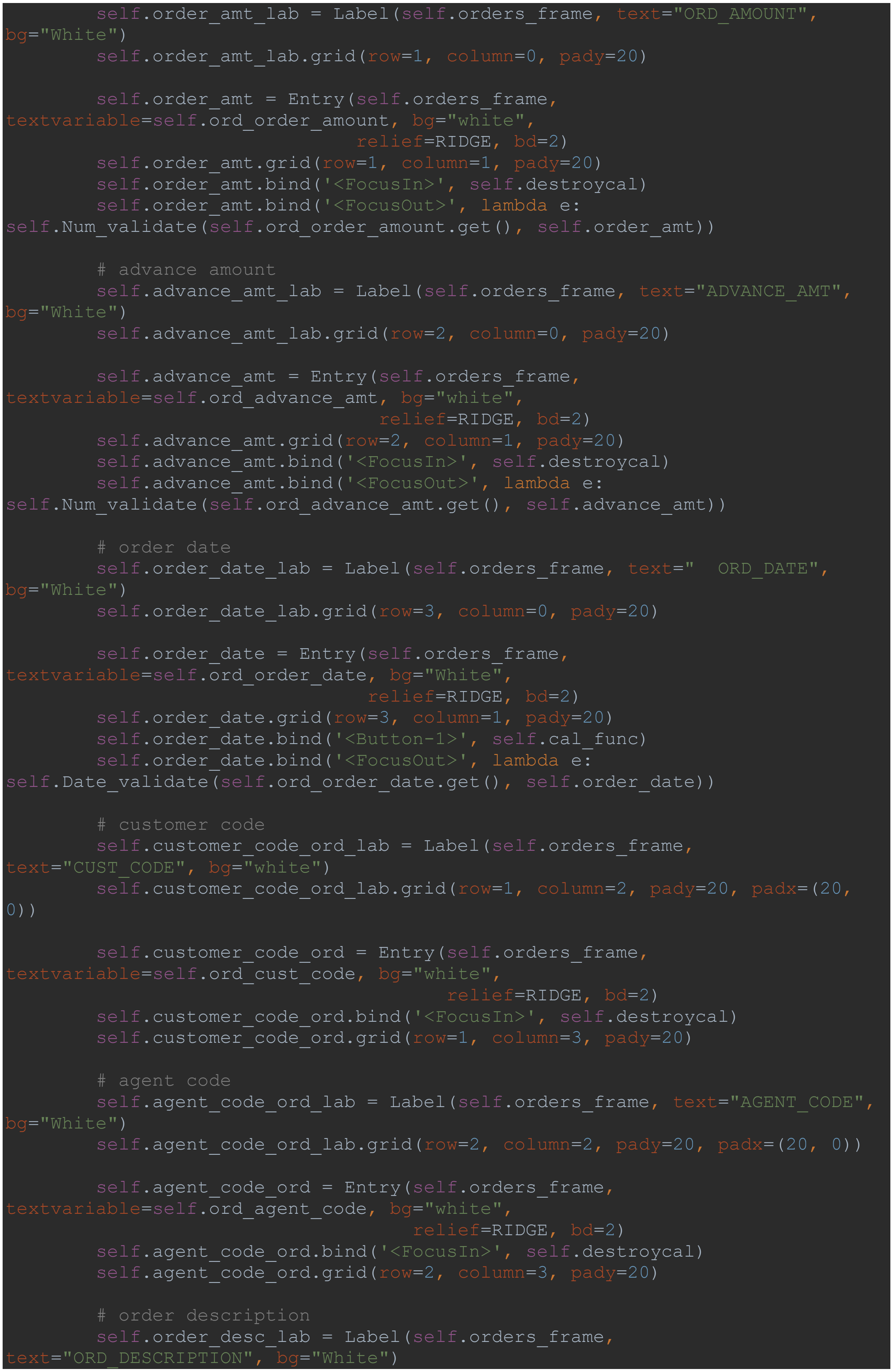


```
        self.table.grid(row=0, column=1, pady=(40, 10))
        self.table.current(0)
        self.table.bind("<<ComboboxSelected>>", self.get_table)

        '''# get table button
        self.get = Button(self.frame1, text="Get Table",
command=self.get_table, bg='black', fg='red',
                          font=("times new roman", 10, "bold"), width=20)
        self.get.grid(row=1, column=0, columnspan=2)'''

        # clear button
        self.clear = Button(self.frame1, text="clear",
command=self.clear_btn, bg="black", fg="red",
                            font=("times new roman", 10, "bold"), width=20)
        self.clear.grid(row=2, column=0, columnspan=2, pady=(10, 0))

        self.frame = Frame(self.frame1, bg="white", width=self.windowWidth /
3 * 2)
        self.frame.grid(row=3, column=0, columnspan=6, pady=20)
        self.agents()

        self.root.mainloop()

    def clear_btn(self):
        for widgets in self.frame.winfo_children():
            widgets.destroy()

    def get_table(self, event=None):

        for widgets in self.frame.winfo_children():
            widgets.destroy()

        self.frame.focus()

        self.table_ch = self.table.get()

        if self.table_ch == "agents":

            self.agents()

        elif self.table_ch == "company":
            self.company()

        elif self.table_ch == "customer":
            self.customer()

        else:
            self.orders()

    def agents(self):
        # defining variables
        self.agent_code = StringVar()
        self.agent_name = StringVar()
        self.working_area = StringVar()
        self.commission = StringVar()
        self.phone_no = StringVar()
        self.country = StringVar()

        self.agent_frame = Frame(self.frame, bg='white')
        self.agent_frame.pack()

        # Agent code entry
        self.code_lab = Label(self.agent_frame, text="AGENT_CODE",
bg="white")
```

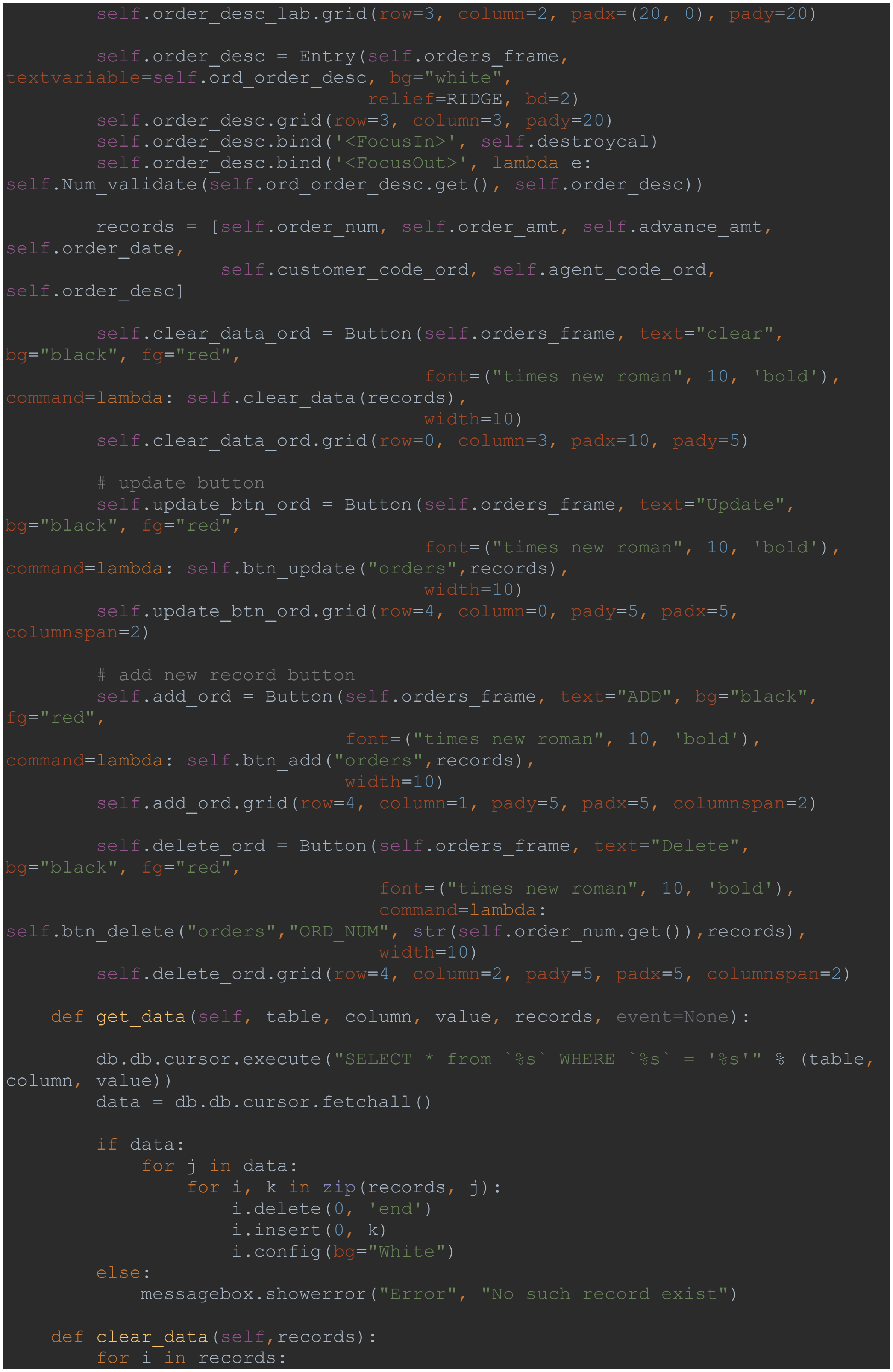


```
        self.code_lab.grid(row=0, column=0, pady=5)

        self.code = Entry(self.agent_frame, textvariable=self.agent_code,
bd=2, relief=RIDGE)
        self.code.focus()
        self.code.bind('<Return>', lambda e: self.get_data("agents",
"AGENT_CODE", str(self.code.get()), variables))
        self.code.grid(row=0, column=1, pady=5)

        # get button
        self.get_val = Button(self.agent_frame, text="Get", bg="black",
fg="red", width=10,
                              command=lambda: self.get_data("agents",
"AGENT_CODE", str(self.code.get()), variables))
        self.get_val.grid(row=0, column=2, padx=10, pady=5)

        # agent name
        self.name_lab = Label(self.agent_frame, text="AGENT_NAME",
bg="white")
        self.name_lab.grid(row=1, column=0, pady=20)

        self.name = Entry(self.agent_frame, textvariable=self.agent_name,
bd=2, relief=RIDGE)
        self.name.grid(row=1, column=1, pady=20)
        self.name.bind('<FocusOut>', lambda e:
self.Str_validate(self.agent_name.get(), self.name))

        # working area
        self.working_lab = Label(self.agent_frame, text="WORKING_AREA",
bg="white")
        self.working_lab.grid(row=2, column=0, pady=20)

        self.working = Entry(self.agent_frame,
textvariable=self.working_area, bd=2, relief=RIDGE)
        self.working.grid(row=2, column=1, pady=20)
        self.working.bind('<FocusOut>', lambda e:
self.Str_validate(self.working_area.get(), self.working))

        # commission
        self.comm_lab = Label(self.agent_frame, text="COMMISSION",
bg="white")
        self.comm_lab.grid(row=3, column=0, pady=20)

        self.comm = Entry(self.agent_frame, textvariable=self.commission,
bd=2, relief=RIDGE)
        self.comm.grid(row=3, column=1, pady=20)
        self.comm.bind('<FocusOut>', lambda e:
self.Num_validate(self.commission.get(), self.comm))

        # phone no
        self.phone_lab = Label(self.agent_frame, text="PHONE_NO", bg="white")
        self.phone_lab.grid(row=1, column=2, pady=20, padx=(20, 0))

        self.phone = Entry(self.agent_frame, textvariable=self.phone_no,
bd=2, relief=RIDGE)
        self.phone.grid(row=1, column=3, pady=20)
        self.phone.bind('<FocusOut>', lambda e:
self.Phone_validate(self.phone_no.get(), self.phone))

        # Country

        self.ctry_lab = Label(self.agent_frame, text="COUNTRY", bg="white")
        self.ctry_lab.grid(row=2, column=2, pady=20, padx=(20, 0))
        self.ctry = Entry(self.agent_frame, textvariable=self.country, bd=2,
relief=RIDGE)
        self.ctry.grid(row=2, column=3, pady=20)
        self.ctry.bind('<FocusOut>', lambda e:
self.Str_validate(self.ctry.get(), self.ctry))

        variables = [self.code, self.name, self.working, self.comm,
self.phone, self.ctry]

        self.clear_data_age = Button(self.agent_frame, text="clear",
bg="black", fg="red",
                                 font=("times new roman", 10, 'bold'),
command=lambda: self.clear_data(variables),
                                 width=10)
        self.clear_data_age.grid(row=0, column=3, padx=10, pady=5)

        # update button
        self.update_btn = Button(self.agent_frame, text="Update", bg="black",
fg="red",
                                 font=("times new roman", 10, 'bold'),
command=lambda: self.btn_update("agents", variables),
                                 width=10)
        self.update_btn.grid(row=4, column=0, pady=5, padx=5, columnspan=2)

        # add new record button
        self.add = Button(self.agent_frame, text="ADD", bg="black", fg="red",
                          font=("times new roman", 10, 'bold'),
command=lambda: self.btn_add("agents", variables),
                          width=10)
        self.add.grid(row=4, column=1, pady=5, padx=5, columnspan=2)

        self.delete_age = Button(self.agent_frame, text="Delete", bg="black",
fg="red",
                                 font=("times new roman", 10, 'bold'),
                                 command=lambda: self.btn_delete("agents",
"AGENT_CODE",

str(self.code.get()),variables), width=10)
        self.delete_age.grid(row=4, column=2, pady=5, padx=5, columnspan=2)

    def company(self):
        # defining variables
        self.company_id = StringVar()
        self.company_name = StringVar()
        self.company_city = StringVar()

        self.company_frame = Frame(self.frame, bg="white")
        self.company_frame.pack()

        # company id
        self.comp_id_lab = Label(self.company_frame, text="COMPANY_ID",
bg="white")
        self.comp_id_lab.grid(row=0, column=0, rowspan=2, pady=5)

        self.comp_id = Entry(self.company_frame,
textvariable=self.company_id, bg="white", relief=RIDGE, bd=2)
        self.comp_id.focus()
        self.comp_id.bind('<Return>', lambda e: self.get_data("company",
"COMPANY_ID", str(self.comp_id.get()),
                                                              records))
        self.comp_id.grid(row=0, column=1, rowspan=2, pady=5)

        # get button
        self.get_val_comp = Button(self.company_frame, text="Get",
bg="black", fg="red", width=10,
                                   command=lambda: self.get_data("company",
"COMPANY_ID", str(self.comp_id.get()),
                                                                 records))
        self.get_val_comp.grid(row=0, column=2, padx=10, pady=5)

        # company name
        self.comp_name_lab = Label(self.company_frame, text="COMPANY_NAME",
bg="white")
        self.comp_name_lab.grid(row=2, column=0, rowspan=2, pady=5, padx=(10,
0))

        self.comp_name = Entry(self.company_frame,
textvariable=self.company_name, bg="white", relief=RIDGE, bd=2)
        self.comp_name.grid(row=2, column=1, rowspan=2, pady=5)
        self.comp_name.bind('<FocusOut>', lambda e:
self.Str_validate(self.company_name.get(), self.comp_name))

        # Company city
        self.comp_city_lab = Label(self.company_frame, text="COMPANY_CITY",
bg="white")
        self.comp_city_lab.grid(row=2, column=2, rowspan=2, pady=5)

        self.comp_city = Entry(self.company_frame,
textvariable=self.company_city, bg="white", relief=RIDGE, bd=2)
        self.comp_city.grid(row=2, column=3, rowspan=2, pady=5)
        self.comp_city.bind('<FocusOut>', lambda e:
self.Str_validate(self.company_city.get(), self.comp_city))

        self.company_frame.rowconfigure(4, minsize=185)

        records = [self.comp_id, self.comp_name, self.comp_city]

        self.clear_data_comp = Button(self.company_frame, text="clear",
bg="black", fg="red",
                                     font=("times new roman", 10, 'bold'),
command=lambda: self.clear_data(records),
                                     width=10)
        self.clear_data_comp.grid(row=0, column=3, padx=10, pady=5)

        # update button
        self.update_btn_comp = Button(self.company_frame, text="Update",
bg="black", fg="red",
                                      font=("times new roman", 10, 'bold'),
command=lambda: self.btn_update('company',records),
                                      width=10)
        self.update_btn_comp.grid(row=4, column=0, pady=5, padx=5,
columnspan=2, sticky=S)

        # add new record button
        self.add_comp = Button(self.company_frame, text="ADD", bg="black",
fg="red",
                               font=("times new roman", 10, 'bold'),
command=lambda: self.btn_add("company", records),
                               width=10)
        self.add_comp.grid(row=4, column=1, pady=5, padx=5, columnspan=2,
sticky=S)

        self.delete_comp = Button(self.company_frame, text="Delete",
bg="black", fg="red",
                                  font=("times new roman", 10, 'bold'),
                                  command=lambda:
self.btn_delete("company","COMPANY_ID",str(self.comp_id.get()),records),
                                  width=10)
        self.delete_comp.grid(row=4, column=2, pady=5, padx=5, columnspan=2,
sticky=S)

    def customer(self):

        # defining variables
        self.cust_code = StringVar()
        self.cust_name = StringVar()
        self.cust_city = StringVar()
        self.cust_working_area = StringVar()
        self.cust_country = StringVar()
        self.cust_grade = StringVar()
        self.cust_opening_amt = StringVar()
        self.cust_receive_amt = StringVar()
        self.cust_payment_amt = StringVar()
        self.cust_outstanding_amt = StringVar()
        self.cust_phone_no = StringVar()
        self.cust_agent_code = StringVar()

        self.customer_frame = Frame(self.frame, bg="white")
        self.customer_frame.pack()

        # customer code
        self.customer_code_lab = Label(self.customer_frame, text="CUST_CODE",
bg="white")
        self.customer_code_lab.grid(row=0, column=0, pady=5)

        self.customer_code = Entry(self.customer_frame,
textvariable=self.cust_code, bg="white", relief=RIDGE, bd=2)
        self.customer_code.focus()
        self.customer_code.bind('<Return>',
                                lambda e: self.get_data("customer",
"CUST_CODE", str(self.customer_code.get()),
                                                        records))
        self.customer_code.grid(row=0, column=1, pady=5)

        # get button
        self.get_val_cust = Button(self.customer_frame, text="Get",
bg="black", fg="red", width=10,
                                   command=lambda: self.get_data("customer",
"CUST_CODE", str(self.customer_code.get()),
                                                                 records))
        self.get_val_cust.grid(row=0, column=2, padx=10, pady=5)

        # customer name

        self.customer_name_lab = Label(self.customer_frame, text="CUST_NAME",
bg="white")
        self.customer_name_lab.grid(row=1, column=0, pady=5)

        self.customer_name = Entry(self.customer_frame,
textvariable=self.cust_name, bg="white", relief=RIDGE, bd=2)
        self.customer_name.grid(row=1, column=1, padx=5)
        self.customer_name.bind('<FocusOut>', lambda e:
self.Str_validate(self.cust_name.get(), self.customer_name))

        # customer city

        self.customer_city_lab = Label(self.customer_frame, text="CUST_CITY",
bg="white")
        self.customer_city_lab.grid(row=2, column=0, pady=5)

        self.customer_city = Entry(self.customer_frame,
textvariable=self.cust_city, bg="white", relief=RIDGE, bd=2)
```

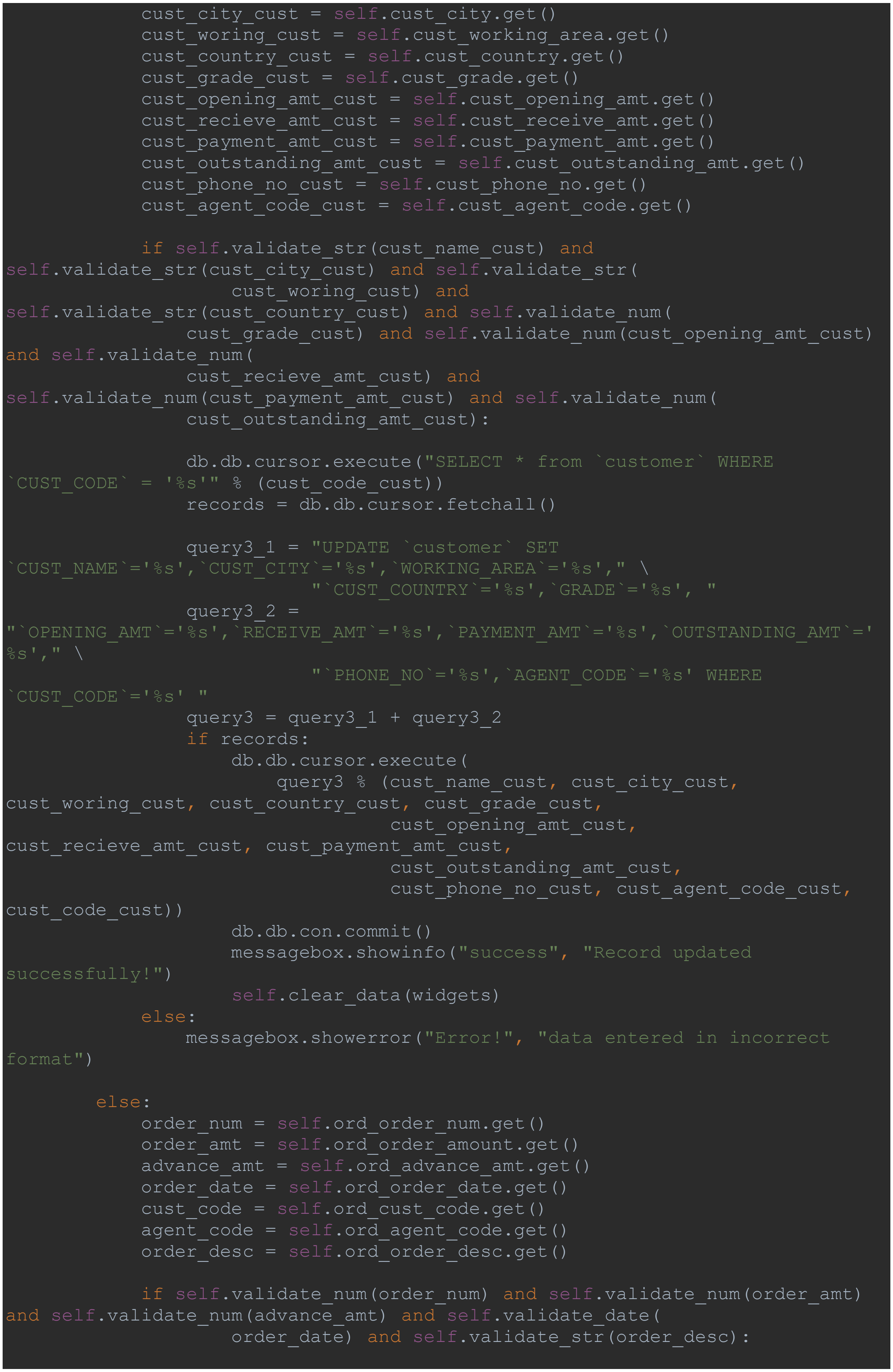
```
        self.customer_city.grid(row=2, column=1, pady=5)
        self.customer_city.bind('<FocusOut>', lambda e:
self.Str_validate(self.cust_city.get(), self.customer_city))

        # working area
        self.working_area_cust_lab = Label(self.customer_frame,
text="WORKING_AREA", bg="white")
        self.working_area_cust_lab.grid(row=3, column=0, pady=5)

        self.working_area_cust = Entry(self.customer_frame,
textvariable=self.cust_working_area, bg="white",
                                       relief=RIDGE, bd=2)
        self.working_area_cust.grid(row=3, column=1, pady=5)
        self.working_area_cust.bind('<FocusOut>',
                                    lambda e:
self.Str_validate(self.cust_working_area.get(), self.working_area_cust))

        # customer country
        self.customer_country_lab = Label(self.customer_frame,
text="CUST_COUNTRY", bg="white")
        self.customer_country_lab.grid(row=4, column=0, pady=5)

        self.customer_country = Entry(self.customer_frame,
textvariable=self.cust_country, bg="white",
                                      relief=RIDGE, bd=2)
        self.customer_country.grid(row=4, column=1, pady=5)
        self.customer_country.bind('<FocusOut>',
                                   lambda e:
self.Str_validate(self.cust_country.get(), self.customer_country))

        # grade
        self.grade_lab = Label(self.customer_frame, text="GRADE", bg="white")
        self.grade_lab.grid(row=5, column=0, pady=5)

        self.grade = Entry(self.customer_frame, textvariable=self.cust_grade,
bg="white", relief=RIDGE, bd=2)
        self.grade.grid(row=5, column=1, padx=5)
        self.grade.bind('<FocusOut>', lambda e:
self.Num_validate(self.cust_grade.get(), self.grade))

        # opening amount
        self.opening_amt_cust_lab = Label(self.customer_frame,
text="OPENING_AMT", bg="white")
        self.opening_amt_cust_lab.grid(row=6, column=0, pady=5)

        self.opening_amt_cust = Entry(self.customer_frame,
textvariable=self.cust_opening_amt, bg="white",
                                      relief=RIDGE, bd=2)
        self.opening_amt_cust.grid(row=6, column=1, pady=5)
        self.opening_amt_cust.bind('<FocusOut>',
                                   lambda e:
self.Num_validate(self.cust_opening_amt.get(), self.opening_amt_cust))

        # recieve amount

        self.receive_amt_cust_lab = Label(self.customer_frame,
text="RECEIVE_AMT", bg="white")
        self.receive_amt_cust_lab.grid(row=1, column=2, padx=(20, 0), pady=5)

        self.receive_amt_cust = Entry(self.customer_frame,
textvariable=self.cust_receive_amt, bg="white",
                                      relief=RIDGE, bd=2)
        self.receive_amt_cust.grid(row=1, column=3, pady=5)
        self.receive_amt_cust.bind('<FocusOut>',
                                   lambda e:
self.Num_validate(self.cust_receive_amt.get(), self.receive_amt_cust))

        # payment amount
        self.payment_amt_cust_lab = Label(self.customer_frame,
text="PAYMENT_AMT", bg="white")
        self.payment_amt_cust_lab.grid(row=2, column=2, padx=(20, 0), pady=5)

        self.payment_amt_cust = Entry(self.customer_frame,
textvariable=self.cust_payment_amt, bg="white",
                                      relief=RIDGE, bd=2)
        self.payment_amt_cust.grid(row=2, column=3, pady=5)
        self.payment_amt_cust.bind('<FocusOut>',
                                   lambda e:
self.Num_validate(self.cust_payment_amt.get(), self.payment_amt_cust))

        # outstanding amt
        self.outstanding_amt_cust_lab = Label(self.customer_frame,
text="OUTSTANDING_AMT", bg="white")
        self.outstanding_amt_cust_lab.grid(row=3, column=2, padx=(20, 0),
pady=5)

        self.outstanding_amt_cust = Entry(self.customer_frame,
textvariable=self.cust_outstanding_amt, bg="white",
                                          relief=RIDGE, bd=2)
        self.outstanding_amt_cust.grid(row=3, column=3, pady=5)
        self.outstanding_amt_cust.bind('<FocusOut>', lambda e:
self.Num_validate(self.cust_outstanding_amt.get(),

self.outstanding_amt_cust))

        # phone no
        self.phone_no_cust_lab = Label(self.customer_frame, text="PHONE_NO",
bg="white")
        self.phone_no_cust_lab.grid(row=4, column=2, padx=(20, 0), pady=5)

        self.phone_no_cust = Entry(self.customer_frame,
textvariable=self.cust_phone_no, bg="white",
                                   relief=RIDGE, bd=2)
        self.phone_no_cust.grid(row=4, column=3, pady=5)


        # agent code

        self.agent_code_cust_lab = Label(self.customer_frame,
text="AGENT_CODE", bg="white")
        self.agent_code_cust_lab.grid(row=5, column=2, padx=(20, 0), pady=5)

        self.agent_code_cust = Entry(self.customer_frame,
textvariable=self.cust_agent_code, bg="white",
                                     relief=RIDGE, bd=2)
        self.agent_code_cust.grid(row=5, column=3, pady=5)

        records = [self.customer_code, self.customer_name,
self.customer_city, self.working_area_cust,
                   self.customer_country,
                   self.grade, self.opening_amt_cust, self.receive_amt_cust,
self.payment_amt_cust,
                   self.outstanding_amt_cust,
                   self.phone_no_cust, self.agent_code_cust]

        self.clear_data_cus = Button(self.customer_frame, text="clear",
bg="black", fg="red",
                                     font=("times new roman", 10, 'bold'),
command=lambda: self.clear_data(records),
                                     width=10)
        self.clear_data_cus.grid(row=0, column=3, padx=10, pady=5)

        # update btn
        self.update_btn_cust = Button(self.customer_frame, text="Update",
bg="black", fg="red",
                                      font=("times new roman", 10, 'bold'),
command=lambda: self.btn_update("customer", records),
                                      width=10)
        self.update_btn_cust.grid(row=7, column=0, pady=5, padx=5,
columnspan=2)

        self.add_cust = Button(self.customer_frame, text="ADD", bg="black",
fg="red",
                               font=("times new roman", 10, 'bold'),
command=lambda: self.btn_add("customer", records),
                               width=10)
        self.add_cust.grid(row=7, column=1, pady=5, padx=5, columnspan=2)
        self.delete_cust = Button(self.customer_frame, text="Delete",
bg="black", fg="red",
                                  font=("times new roman", 10, 'bold'),
                                  command=lambda:
self.btn_delete('customer','CUST_CODE',str(self.customer_code), records),
                                  width=10)
        self.delete_cust.grid(row=7, column=2, pady=5, padx=5, columnspan=2)

    def orders(self):
        # setting variables
        self.ord_order_num = StringVar()
        self.ord_order_amount = StringVar()
        self.ord_advance_amt = StringVar()
        self.ord_order_date = StringVar()
        self.ord_cust_code = StringVar()
        self.ord_agent_code = StringVar()
        self.ord_order_desc = StringVar()

        self.orders_frame = Frame(self.frame, bg="White")
        self.orders_frame.pack()

        self.order_num_lab = Label(self.orders_frame, text="ORD_NUM",
bg="White")
        self.order_num_lab.grid(row=0, column=0, pady=5)

        self.order_num = Entry(self.orders_frame,
textvariable=self.ord_order_num, bg="white",
                               relief=RIDGE, bd=2)
        self.order_num.focus()
        self.order_num.bind('<Return>', lambda e: self.get_data("orders",
"ORD_NUM", str(self.order_num.get()),
                                                                records))
        self.order_num.grid(row=0, column=1, pady=5)
        self.order_num.bind('<FocusOut>', lambda e:
self.Num_validate(self.ord_order_num.get(), self.order_num))

        # get button
        self.get_val_ord = Button(self.orders_frame, text="Get", bg="black",
fg="red", width=10,
                                  command=lambda: self.get_data("orders",
"ORD_NUM", str(self.order_num.get()),
                                                                records))
        self.get_val_ord.grid(row=0, column=2, padx=10, pady=5)

        # order amount
```



```
        self.order_amt_lab = Label(self.orders_frame, text="ORD_AMOUNT",
bg="White")
        self.order_amt_lab.grid(row=1, column=0, pady=20)

        self.order_amt = Entry(self.orders_frame,
textvariable=self.ord_order_amount, bg="white",
                               relief=RIDGE, bd=2)
        self.order_amt.grid(row=1, column=1, pady=20)
        self.order_amt.bind('<FocusIn>', self.destroycal)
        self.order_amt.bind('<FocusOut>', lambda e:
self.Num_validate(self.ord_order_amount.get(), self.order_amt))

        # advance amount
        self.advance_amt_lab = Label(self.orders_frame, text="ADVANCE_AMT",
bg="White")
        self.advance_amt_lab.grid(row=2, column=0, pady=20)

        self.advance_amt = Entry(self.orders_frame,
textvariable=self.ord_advance_amt, bg="white",
                                 relief=RIDGE, bd=2)
        self.advance_amt.grid(row=2, column=1, pady=20)
        self.advance_amt.bind('<FocusIn>', self.destroycal)
        self.advance_amt.bind('<FocusOut>', lambda e:
self.Num_validate(self.ord_advance_amt.get(), self.advance_amt))

        # order date
        self.order_date_lab = Label(self.orders_frame, text="  ORD_DATE",
bg="White")
        self.order_date_lab.grid(row=3, column=0, pady=20)

        self.order_date = Entry(self.orders_frame,
textvariable=self.ord_order_date, bg="White",
                                relief=RIDGE, bd=2)
        self.order_date.grid(row=3, column=1, pady=20)
        self.order_date.bind('<Button-1>', self.cal_func)
        self.order_date.bind('<FocusOut>', lambda e:
self.Date_validate(self.ord_order_date.get(), self.order_date))

        # customer code
        self.customer_code_ord_lab = Label(self.orders_frame,
text="CUST_CODE", bg="white")
        self.customer_code_ord_lab.grid(row=1, column=2, pady=20, padx=(20,
0))

        self.customer_code_ord = Entry(self.orders_frame,
textvariable=self.ord_cust_code, bg="white",
                                       relief=RIDGE, bd=2)
        self.customer_code_ord.bind('<FocusIn>', self.destroycal)
        self.customer_code_ord.grid(row=1, column=3, pady=20)

        # agent code
        self.agent_code_ord_lab = Label(self.orders_frame, text="AGENT_CODE",
bg="White")
        self.agent_code_ord_lab.grid(row=2, column=2, pady=20, padx=(20, 0))

        self.agent_code_ord = Entry(self.orders_frame,
textvariable=self.ord_agent_code, bg="white",
                                    relief=RIDGE, bd=2)
        self.agent_code_ord.bind('<FocusIn>', self.destroycal)
        self.agent_code_ord.grid(row=2, column=3, pady=20)

        # order description
        self.order_desc_lab = Label(self.orders_frame,
text="ORD_DESCRIPTION", bg="White")
```

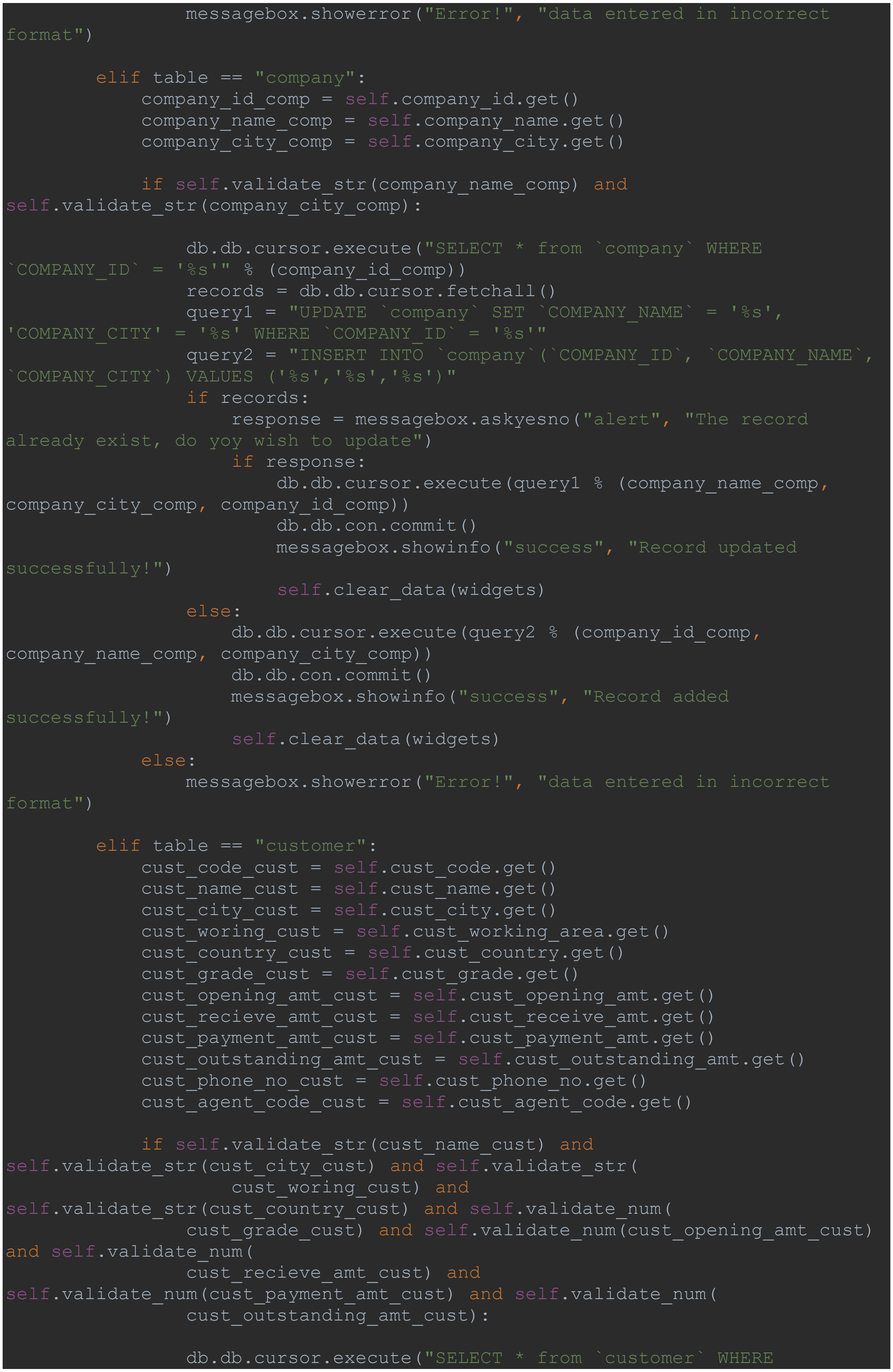


```
        self.order_desc_lab.grid(row=3, column=2, padx=(20, 0), pady=20)

        self.order_desc = Entry(self.orders_frame,
textvariable=self.ord_order_desc, bg="white",
                                relief=RIDGE, bd=2)
        self.order_desc.grid(row=3, column=3, pady=20)
        self.order_desc.bind('<FocusIn>', self.destroycal)
        self.order_desc.bind('<FocusOut>', lambda e:
self.Num_validate(self.ord_order_desc.get(), self.order_desc))

        records = [self.order_num, self.order_amt, self.advance_amt,
self.order_date,
                   self.customer_code_ord, self.agent_code_ord,
self.order_desc]

        self.clear_data_ord = Button(self.orders_frame, text="clear",
bg="black", fg="red",
                                     font=("times new roman", 10, 'bold'),
command=lambda: self.clear_data(records),
                                     width=10)
        self.clear_data_ord.grid(row=0, column=3, padx=10, pady=5)

        # update button
        self.update_btn_ord = Button(self.orders_frame, text="Update",
bg="black", fg="red",
                                     font=("times new roman", 10, 'bold'),
command=lambda: self.btn_update("orders",records),
                                     width=10)
        self.update_btn_ord.grid(row=4, column=0, pady=5, padx=5,
columnspan=2)

        # add new record button
        self.add_ord = Button(self.orders_frame, text="ADD", bg="black",
fg="red",
                              font=("times new roman", 10, 'bold'),
command=lambda: self.btn_add("orders",records),
                              width=10)
        self.add_ord.grid(row=4, column=1, pady=5, padx=5, columnspan=2)

        self.delete_ord = Button(self.orders_frame, text="Delete",
bg="black", fg="red",
                                 font=("times new roman", 10, 'bold'),
                                 command=lambda:
self.btn_delete("orders","ORD_NUM", str(self.order_num.get()),records),
                                 width=10)
        self.delete_ord.grid(row=4, column=2, pady=5, padx=5, columnspan=2)

    def get_data(self, table, column, value, records, event=None):

        db.db.cursor.execute("SELECT * from `%s` WHERE `%s` = '%s'" % (table,
column, value))
        data = db.db.cursor.fetchall()

        if data:
            for j in data:
                for i, k in zip(records, j):
                    i.delete(0, 'end')
                    i.insert(0, k)
                    i.config(bg="White")
        else:
            messagebox.showerror("Error", "No such record exist")

    def clear_data(self,records):
        for i in records:
            i.delete(0,'end')
        records[0].focus()

    def btn_update(self, table, widgets):

        if table == "agents":
            agent_code_age = self.agent_code.get()
            agent_name_age = self.agent_name.get()
            working_area_age = self.working_area.get()
            commission_age = self.commission.get()
            phone_no_age = self.phone_no.get()
            country_age = self.country.get()

            if self.validate_str(agent_name_age) and
self.validate_str(working_area_age) and self.validate_num(
                    commission_age) and self.validate_phone(phone_no_age) and
self.validate_str(country_age):

                db.db.cursor.execute("SELECT * from `agents` WHERE
`AGENT_CODE` = '%s'" % (agent_code_age))
                records = db.db.cursor.fetchall()
                query1 = "UPDATE `agents` SET
`AGENT_NAME`='%s',`WORKING_AREA`='%s',`COMMISSION`='%s'," \
                         "`PHONE_NO`='%s',`COUNTRY`='%s' WHERE
`AGENT_CODE`='%s' "
                if records:
                    db.db.cursor.execute(query1 % (
                        agent_name_age, working_area_age, commission_age,
phone_no_age, country_age, agent_code_age))
                    db.db.con.commit()
                    messagebox.showinfo("success", "Record updated
successfully!")
                    self.clear_data(widgets)
            else:
                messagebox.showerror("Error!", "data entered in incorrect
format")

        elif table == "company":
            company_id_comp = self.company_id.get()
            company_name_comp = self.company_name.get()
            company_city_comp = self.company_city.get()

            if self.validate_str(company_name_comp) and
self.validate_str(company_city_comp):

                db.db.cursor.execute("SELECT * from `company` WHERE
`COMPANY_ID` = '%s'" % company_id_comp)
                records = db.db.cursor.fetchall()
                query2 = "UPDATE `company` SET `COMPANY_NAME` = '%s',
`COMPANY_CITY` = '%s' WHERE `COMPANY_ID` = '%s'"
                if records:
                    db.db.cursor.execute(query2 % (company_name_comp,
company_city_comp, company_id_comp))
                    db.db.con.commit()
                    messagebox.showinfo("success", "Record updated
successfully!")
                    self.clear_data(widgets)
            else:
                messagebox.showerror("Error!", "data entered in incorrect
format")

        elif table == "customer":
            cust_code_cust = self.cust_code.get()
            cust_name_cust = self.cust_name.get()
```

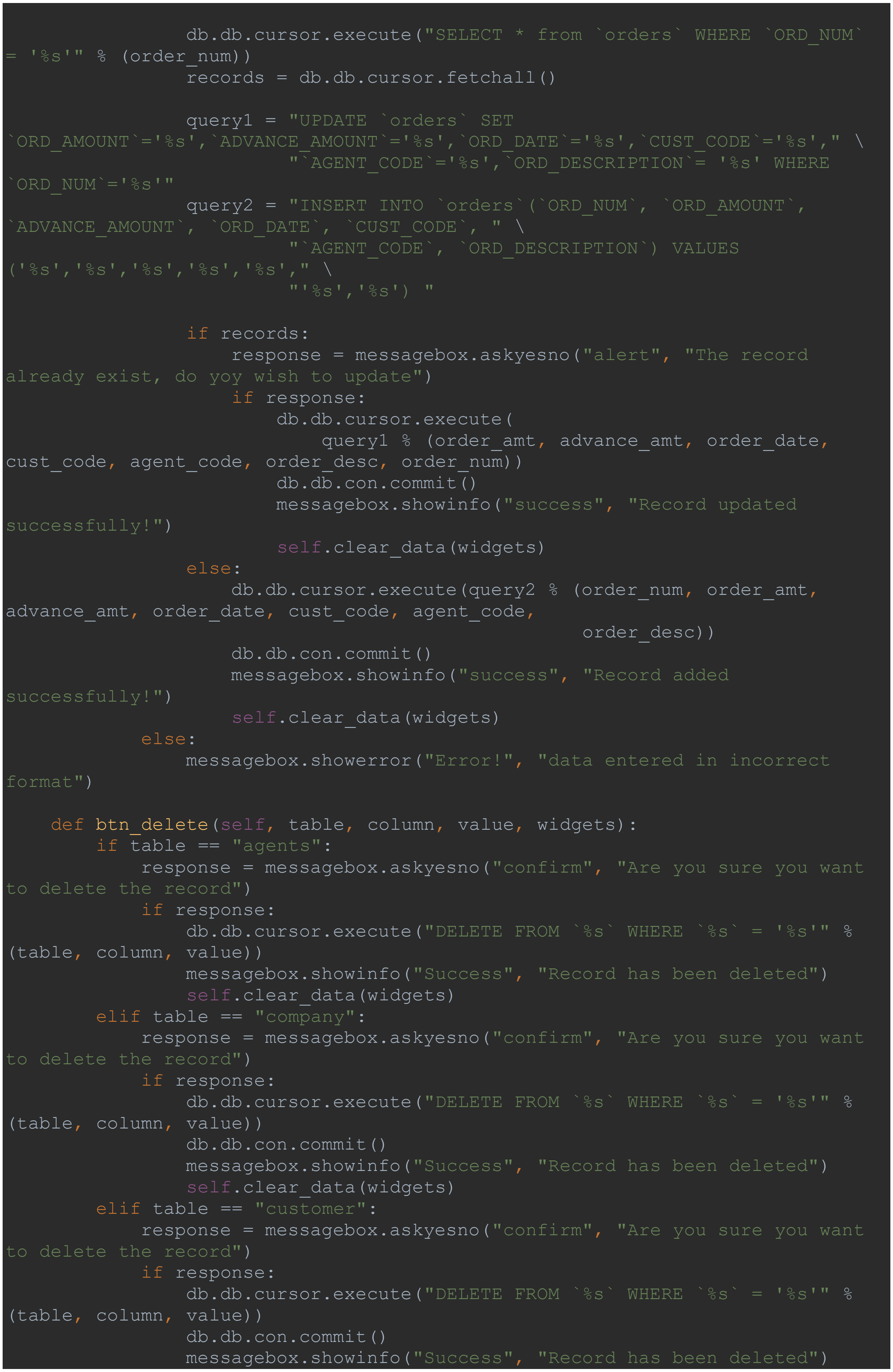


```
            cust_city_cust = self.cust_city.get()
            cust_woring_cust = self.cust_working_area.get()
            cust_country_cust = self.cust_country.get()
            cust_grade_cust = self.cust_grade.get()
            cust_opening_amt_cust = self.cust_opening_amt.get()
            cust_recieve_amt_cust = self.cust_receive_amt.get()
            cust_payment_amt_cust = self.cust_payment_amt.get()
            cust_outstanding_amt_cust = self.cust_outstanding_amt.get()
            cust_phone_no_cust = self.cust_phone_no.get()
            cust_agent_code_cust = self.cust_agent_code.get()

            if self.validate_str(cust_name_cust) and
self.validate_str(cust_city_cust) and self.validate_str(
                    cust_woring_cust) and
self.validate_str(cust_country_cust) and self.validate_num(
                cust_grade_cust) and self.validate_num(cust_opening_amt_cust)
and self.validate_num(
                cust_recieve_amt_cust) and
self.validate_num(cust_payment_amt_cust) and self.validate_num(
                cust_outstanding_amt_cust):

                db.db.cursor.execute("SELECT * from `customer` WHERE
`CUST_CODE` = '%s'" % (cust_code_cust))
                records = db.db.cursor.fetchall()

                query3_1 = "UPDATE `customer` SET
`CUST_NAME`='%s',`CUST_CITY`='%s',`WORKING_AREA`='%s'," \
                           "`CUST_COUNTRY`='%s',`GRADE`='%s', "
                query3_2 =
"`OPENING_AMT`='%s',`RECEIVE_AMT`='%s',`PAYMENT_AMT`='%s',`OUTSTANDING_AMT`='
%s'," \
                           "`PHONE_NO`='%s',`AGENT_CODE`='%s' WHERE
`CUST_CODE`='%s' "
                query3 = query3_1 + query3_2
                if records:
                    db.db.cursor.execute(
                        query3 % (cust_name_cust, cust_city_cust,
cust_woring_cust, cust_country_cust, cust_grade_cust,
                                  cust_opening_amt_cust,
cust_recieve_amt_cust, cust_payment_amt_cust,
                                  cust_outstanding_amt_cust,
                                  cust_phone_no_cust, cust_agent_code_cust,
cust_code_cust))
                    db.db.con.commit()
                    messagebox.showinfo("success", "Record updated
successfully!")
                    self.clear_data(widgets)
            else:
                messagebox.showerror("Error!", "data entered in incorrect
format")

        else:
            order_num = self.ord_order_num.get()
            order_amt = self.ord_order_amount.get()
            advance_amt = self.ord_advance_amt.get()
            order_date = self.ord_order_date.get()
            cust_code = self.ord_cust_code.get()
            agent_code = self.ord_agent_code.get()
            order_desc = self.ord_order_desc.get()

            if self.validate_num(order_num) and self.validate_num(order_amt)
and self.validate_num(advance_amt) and self.validate_date(
                    order_date) and self.validate_str(order_desc):
```

```
                db.db.cursor.execute("SELECT * from `orders` WHERE `ORD_NUM`
= '%s'" % (order_num))
                records = db.db.cursor.fetchall()

                query4 = "UPDATE `orders` SET
`ORD_AMOUNT`='%s',`ADVANCE_AMOUNT`='%s',`ORD_DATE`='%s',`CUST_CODE`='%s'," \
                         "`AGENT_CODE`='%s',`ORD_DESCRIPTION`= '%s' WHERE
`ORD_NUM`='%s'"

                if records:
                    db.db.cursor.execute(
                        query4 % (order_amt, advance_amt, order_date,
cust_code, agent_code, order_desc, order_num))
                    db.db.con.commit()
                    messagebox.showinfo("success", "Record updated
successfully!")
                    self.clear_data(widgets)
            else:
                messagebox.showerror("Error!", "data entered in incorrect
format")

    def btn_add(self, table, widgets):
        if table == "agents":
            agent_code_age = self.agent_code.get()
            agent_name_age = self.agent_name.get()
            working_area_age = self.working_area.get()
            commission_age = self.commission.get()
            phone_no_age = self.phone_no.get()
            country_age = self.country.get()
            if self.validate_str(agent_name_age) and
self.validate_str(working_area_age) and self.validate_num(
                    commission_age) and self.validate_phone(phone_no_age) and
self.validate_str(country_age):

                db.db.cursor.execute("SELECT * from `agents` WHERE
`AGENT_CODE` = '%s'" % (agent_code_age))
                records = db.db.cursor.fetchall()
                query1 = "UPDATE `agents` SET
`AGENT_NAME`='%s',`WORKING_AREA`='%s',`COMMISSION`='%s',`PHONE_NO`='%s',`COUN
TRY`='%s' WHERE `AGENT_CODE`='%s'"
                query2 = "INSERT INTO `agents`(`AGENT_CODE`, `AGENT_NAME`,
`WORKING_AREA`, `COMMISSION`, `PHONE_NO`, `COUNTRY`) VALUES " \
                         "('%s','%s','%s','%s','%s','%s')"
                if records:
                    response = messagebox.askyesno("alert", "The record
already exist, do yoy wish to update")
                    if response:
                        db.db.cursor.execute(query1 % (
                            agent_name_age, working_area_age, commission_age,
phone_no_age, country_age,
                            agent_code_age))
                        db.db.con.commit()
                        messagebox.showinfo("success", "Record updated
successfully!")
                        self.clear_data(widgets)
                else:
                    db.db.cursor.execute(query2 % (
                        agent_code_age, agent_name_age, working_area_age,
commission_age, phone_no_age, country_age))
                    db.db.con.commit()
                    messagebox.showinfo("success", "Record added
successfully!")
                    self.clear_data(widgets)
            else:
```



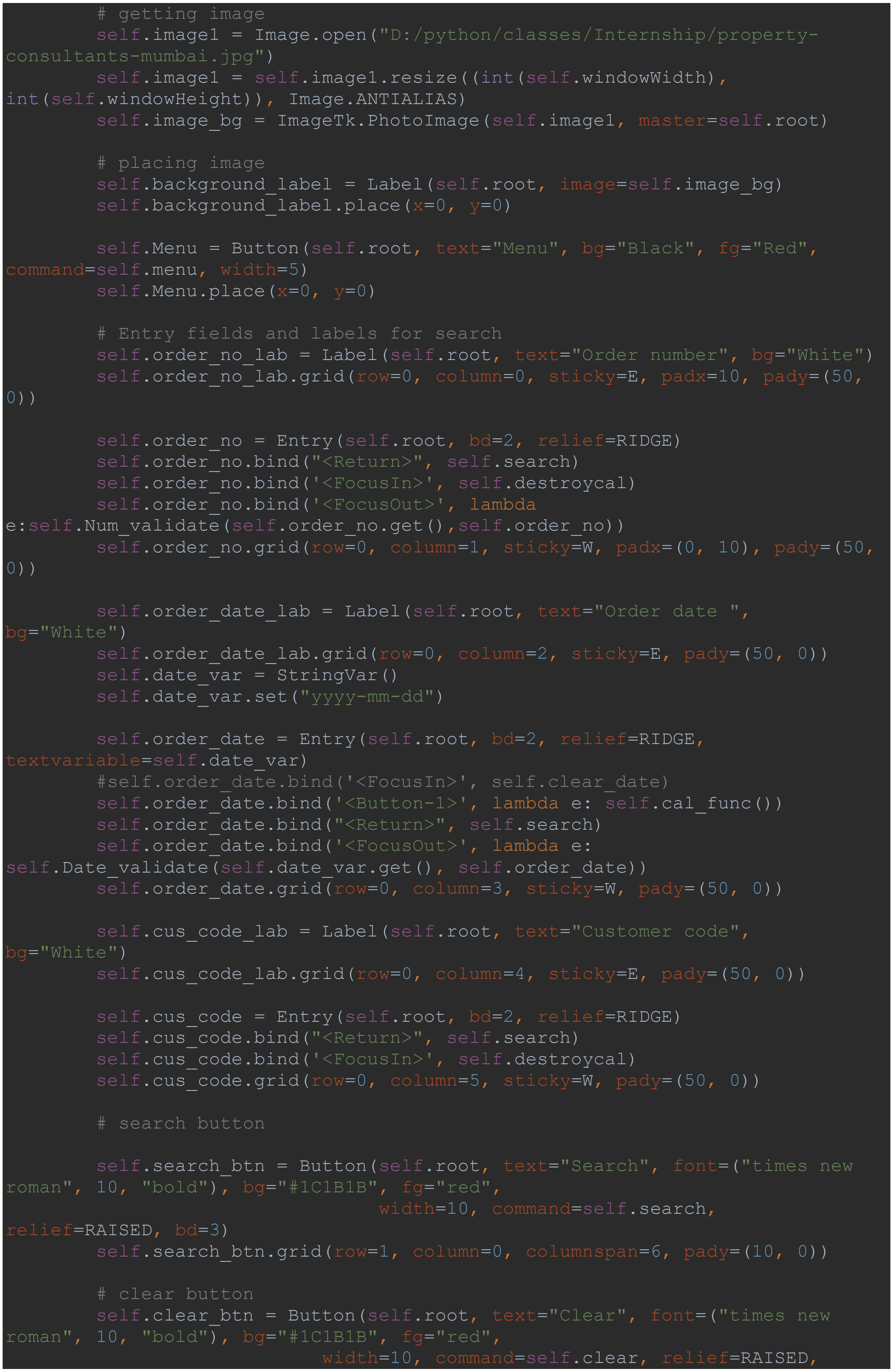
```
                messagebox.showerror("Error!", "data entered in incorrect
format")

        elif table == "company":
            company_id_comp = self.company_id.get()
            company_name_comp = self.company_name.get()
            company_city_comp = self.company_city.get()

            if self.validate_str(company_name_comp) and
self.validate_str(company_city_comp):

                db.db.cursor.execute("SELECT * from `company` WHERE
`COMPANY_ID` = '%s'" % (company_id_comp))
                records = db.db.cursor.fetchall()
                query1 = "UPDATE `company` SET `COMPANY_NAME` = '%s',
'COMPANY_CITY' = '%s' WHERE `COMPANY_ID` = '%s'"
                query2 = "INSERT INTO `company`(`COMPANY_ID`, `COMPANY_NAME`,
`COMPANY_CITY`) VALUES ('%s','%s','%s')"
                if records:
                    response = messagebox.askyesno("alert", "The record
already exist, do yoy wish to update")
                    if response:
                        db.db.cursor.execute(query1 % (company_name_comp,
company_city_comp, company_id_comp))
                        db.db.con.commit()
                        messagebox.showinfo("success", "Record updated
successfully!")
                        self.clear_data(widgets)
                else:
                    db.db.cursor.execute(query2 % (company_id_comp,
company_name_comp, company_city_comp))
                    db.db.con.commit()
                    messagebox.showinfo("success", "Record added
successfully!")
                    self.clear_data(widgets)
            else:
                messagebox.showerror("Error!", "data entered in incorrect
format")

        elif table == "customer":
            cust_code_cust = self.cust_code.get()
            cust_name_cust = self.cust_name.get()
            cust_city_cust = self.cust_city.get()
            cust_woring_cust = self.cust_working_area.get()
            cust_country_cust = self.cust_country.get()
            cust_grade_cust = self.cust_grade.get()
            cust_opening_amt_cust = self.cust_opening_amt.get()
            cust_recieve_amt_cust = self.cust_receive_amt.get()
            cust_payment_amt_cust = self.cust_payment_amt.get()
            cust_outstanding_amt_cust = self.cust_outstanding_amt.get()
            cust_phone_no_cust = self.cust_phone_no.get()
            cust_agent_code_cust = self.cust_agent_code.get()

            if self.validate_str(cust_name_cust) and
self.validate_str(cust_city_cust) and self.validate_str(
                    cust_woring_cust) and
self.validate_str(cust_country_cust) and self.validate_num(
                cust_grade_cust) and self.validate_num(cust_opening_amt_cust)
and self.validate_num(
                cust_recieve_amt_cust) and
self.validate_num(cust_payment_amt_cust) and self.validate_num(
                cust_outstanding_amt_cust):

                db.db.cursor.execute("SELECT * from `customer` WHERE
`CUST_CODE` = '%s'" % (cust_code_cust))
                records = db.db.cursor.fetchall()

                query3_1 = "UPDATE `customer` SET
`CUST_NAME`='%s',`CUST_CITY`='%s',`WORKING_AREA`='%s'," \
                           "`CUST_COUNTRY`='%s',`GRADE`='%s', "
                query3_2 =
"`OPENING_AMT`='%s',`RECEIVE_AMT`='%s',`PAYMENT_AMT`='%s',`OUTSTANDING_AMT`='
%s'," \
                           "`PHONE_NO`='%s',`AGENT_CODE`='%s' WHERE
`CUST_CODE`='%s' "
                query1 = query3_1 + query3_2
                query2 = "INSERT INTO `customer`(`CUST_CODE`, `CUST_NAME`,
`CUST_CITY`, `WORKING_AREA`, `CUST_COUNTRY`, " \
                         "`GRADE`, `OPENING_AMT`, `RECEIVE_AMT`,
`PAYMENT_AMT`, `OUTSTANDING_AMT`, `PHONE_NO`, " \
                         "`AGENT_CODE`) VALUES
('%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s') "
                if records:
                    response = messagebox.askyesno("alert", "The record
already exist, do yoy wish to update")
                    if response:
                        db.db.cursor.execute(
                            query1 % (
                                cust_name_cust, cust_city_cust,
cust_woring_cust, cust_country_cust, cust_grade_cust,
                                cust_opening_amt_cust, cust_recieve_amt_cust,
cust_payment_amt_cust,
                                cust_outstanding_amt_cust,
                                cust_phone_no_cust, cust_agent_code_cust,
cust_code_cust))
                        db.db.con.commit()
                        messagebox.showinfo("success", "Record updated
successfully!")
                        self.clear_data(widgets)

                else:
                    db.db.cursor.execute(query2 % (
                        cust_code_cust, cust_name_cust, cust_city_cust,
cust_woring_cust, cust_country_cust,
                        cust_grade_cust,
                        cust_opening_amt_cust, cust_recieve_amt_cust,
cust_payment_amt_cust, cust_outstanding_amt_cust,
                        cust_phone_no_cust, cust_agent_code_cust))
                    db.db.con.commit()
                    messagebox.showinfo("success", "Record added
successfully!")
                    self.clear_data(widgets)
            else:
                messagebox.showerror("Error!", "data entered in incorrect
format")

        else:
            order_num = self.ord_order_num.get()
            order_amt = self.ord_order_amount.get()
            advance_amt = self.ord_advance_amt.get()
            order_date = self.ord_order_date.get()
            cust_code = self.ord_cust_code.get()
            agent_code = self.ord_agent_code.get()
            order_desc = self.ord_order_desc.get()

            if self.validate_num(order_num) and self.validate_num(order_amt)
and self.validate_num(advance_amt) and self.validate_date(
                    order_date) and self.validate_str(order_desc):
```



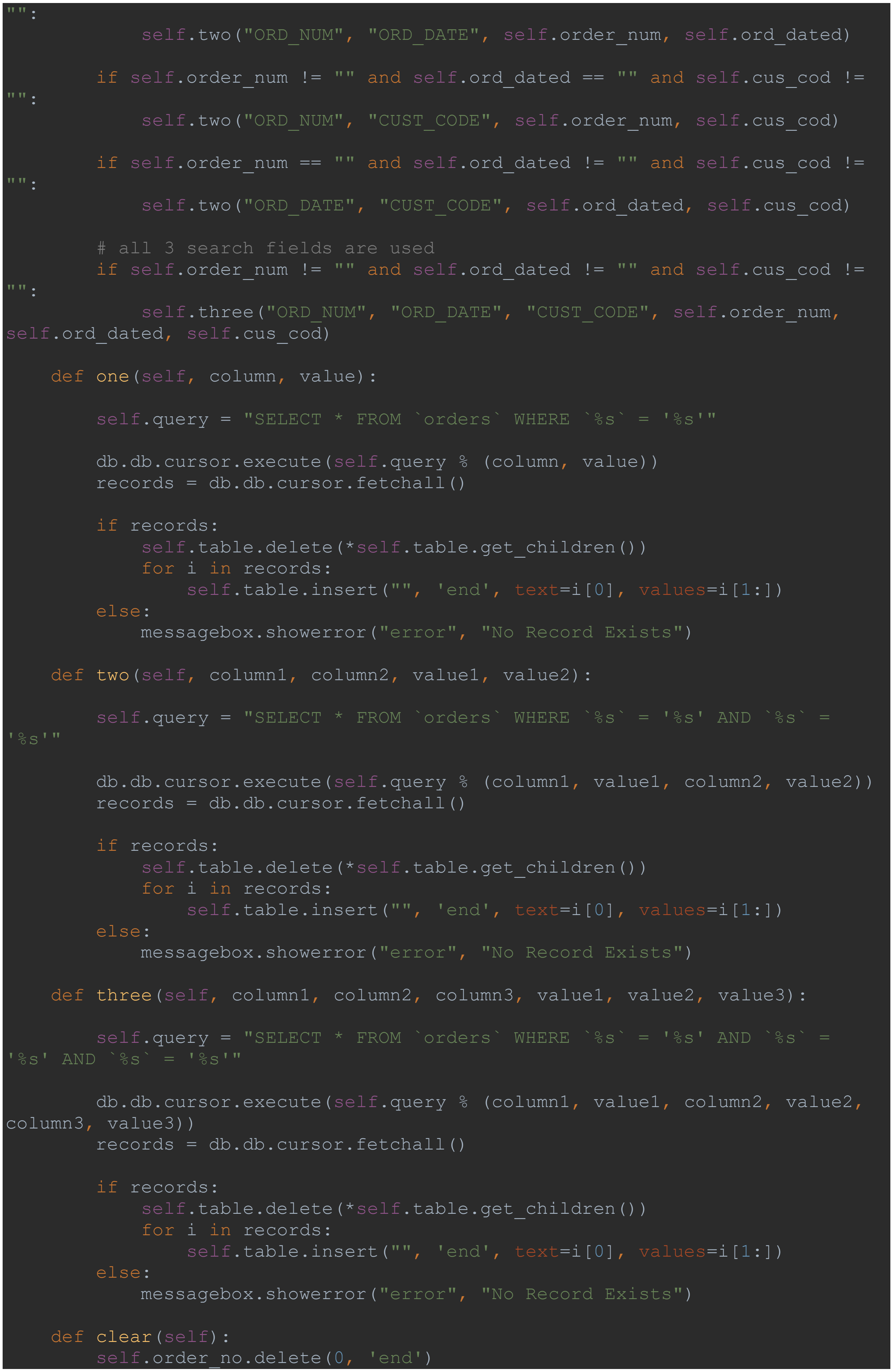
```
                db.db.cursor.execute("SELECT * from `orders` WHERE `ORD_NUM`
= '%s'" % (order_num))
                records = db.db.cursor.fetchall()

                query1 = "UPDATE `orders` SET
`ORD_AMOUNT`='%s',`ADVANCE_AMOUNT`='%s',`ORD_DATE`='%s',`CUST_CODE`='%s'," \
                         "`AGENT_CODE`='%s',`ORD_DESCRIPTION`= '%s' WHERE
`ORD_NUM`='%s'"
                query2 = "INSERT INTO `orders`(`ORD_NUM`, `ORD_AMOUNT`,
`ADVANCE_AMOUNT`, `ORD_DATE`, `CUST_CODE`, " \
                         "`AGENT_CODE`, `ORD_DESCRIPTION`) VALUES
('%s','%s','%s','%s','%s'," \
                         "'%s','%s') "

                if records:
                    response = messagebox.askyesno("alert", "The record
already exist, do yoy wish to update")
                    if response:
                        db.db.cursor.execute(
                            query1 % (order_amt, advance_amt, order_date,
cust_code, agent_code, order_desc, order_num))
                        db.db.con.commit()
                        messagebox.showinfo("success", "Record updated
successfully!")
                        self.clear_data(widgets)
                else:
                    db.db.cursor.execute(query2 % (order_num, order_amt,
advance_amt, order_date, cust_code, agent_code,
                                                   order_desc))
                    db.db.con.commit()
                    messagebox.showinfo("success", "Record added
successfully!")
                    self.clear_data(widgets)
            else:
                messagebox.showerror("Error!", "data entered in incorrect
format")

    def btn_delete(self, table, column, value, widgets):
        if table == "agents":
            response = messagebox.askyesno("confirm", "Are you sure you want
to delete the record")
            if response:
                db.db.cursor.execute("DELETE FROM `%s` WHERE `%s` = '%s'" %
(table, column, value))
                messagebox.showinfo("Success", "Record has been deleted")
                self.clear_data(widgets)
        elif table == "company":
            response = messagebox.askyesno("confirm", "Are you sure you want
to delete the record")
            if response:
                db.db.cursor.execute("DELETE FROM `%s` WHERE `%s` = '%s'" %
(table, column, value))
                db.db.con.commit()
                messagebox.showinfo("Success", "Record has been deleted")
                self.clear_data(widgets)
        elif table == "customer":
            response = messagebox.askyesno("confirm", "Are you sure you want
to delete the record")
            if response:
                db.db.cursor.execute("DELETE FROM `%s` WHERE `%s` = '%s'" %
(table, column, value))
                db.db.con.commit()
                messagebox.showinfo("Success", "Record has been deleted")
                self.clear_data(widgets)
        else:
            response = messagebox.askyesno("confirm", "Are you sure you want
to delete the record")
            if response:
                db.db.cursor.execute("DELETE FROM `%s` WHERE `%s` = '%s'" %
(table, column, value))
                db.db.con.commit()
                messagebox.showinfo("Success", "Record has been deleted")
                self.clear_data(widgets)

    def menu(self):
        self.root.destroy()
        x = menu.Menu()
        x.menu()


if __name__ == "__main__":
    x = Update()
    x.update()
```

Orders lookup Module:

```
from tkinter import *
from PIL import ImageTk, Image
from tkinter import messagebox
from tkinter import ttk
from tkcalendar import *
from datetime import *
import db.db
import menu


class orders_lookup:
    def __init__(self):

        # creating tkinter window
        self.root = Tk()
        self.top = Toplevel(self.root)
        self.top.destroy()

        # Setting title
        self.root.title("Sunville Properties | Orders")

        # determining size of the window
        self.windowWidth = self.root.winfo_screenwidth() / 2
        self.windowHeight = self.root.winfo_screenheight() / 2

        # determining the the positon to set the window
        self.positionRight = int(self.root.winfo_screenwidth() / 2 -
self.windowWidth / 2)
        self.positionDown = int(self.root.winfo_screenheight() / 2 -
self.windowHeight / 2)

        # Positions the window in the center of the page.
        self.root.geometry(
            "{}x{}+{}+{}".format(int(self.windowWidth),
int(self.windowHeight), self.positionRight, self.positionDown))

        # disable resize of window
        self.root.resizable(False, False)
        self.date = 1
        self.firstclick = 1
    def validate_num(self, number):
        try:
            float('%s' % number)
            # messagebox.showinfo("ok", "number")
            return True
        except ValueError:
            return False
    def Num_validate(self, num, widget, event=None):
        if self.validate_num(num) or num == "":
            widget.config(bg="White")
        else:
            widget.config(bg="Red")

    def validate_date(self, date):
        if date=='yyyy-mm-dd':
            return True
        else:
            try:
                datetime.strptime(date, '%Y-%m-%d')
                return True
            except ValueError:
                return False

    def Date_validate(self, date, widget, event=None):
        if self.validate_date(date) or date == "":
            widget.config(bg="White")
        else:
            widget.config(bg="Red")

    def cal_func(self, event=None):
        def calval(event=None):
            self.date_var.set(cal.get_date())
            self.top.destroy()
            self.order_date.config(bg="White")
            self.date = 1

        if self.firstclick == 1:
            self.date_var.set("")
            self.firstclick = 2

        if self.date == 1:
            self.top = Toplevel(self.root)
            # Positions the window in the center of the page.
            self.top.geometry(
                "+{0}+{1}".format(self.positionRight,
                                  self.positionDown))
            cal = Calendar(self.top, font="Arial 14", selectmode="day",
year=datetime.today().year,
                           month=datetime.today().month,
day=datetime.today().day, date_pattern='yyyy-mm-dd')
            cal.pack()
            btn = Button(self.top, text="Ok", command=calval)
            btn.pack()
            self.date = 2

    def destroycal(self, event=None):
        if self.top.winfo_exists():
            self.top.destroy()
            self.date = 1

    def orders(self):

        self.query = "SELECT * from `orders` WHERE 1"
```



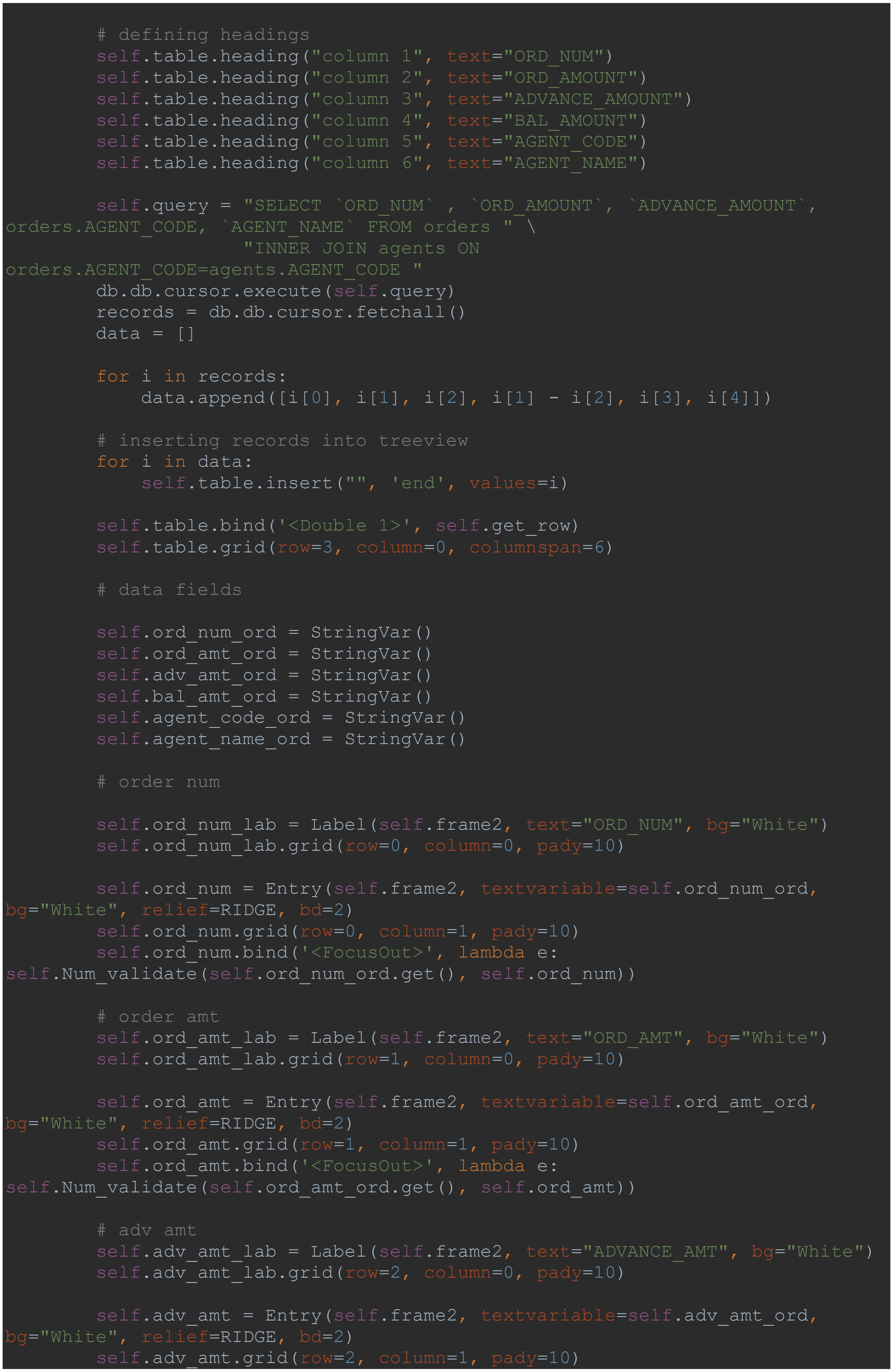
```
        # getting image
        self.image1 = Image.open("D:/python/classes/Internship/property-
consultants-mumbai.jpg")
        self.image1 = self.image1.resize((int(self.windowWidth),
int(self.windowHeight)), Image.ANTIALIAS)
        self.image_bg = ImageTk.PhotoImage(self.image1, master=self.root)

        # placing image
        self.background_label = Label(self.root, image=self.image_bg)
        self.background_label.place(x=0, y=0)

        self.Menu = Button(self.root, text="Menu", bg="Black", fg="Red",
command=self.menu, width=5)
        self.Menu.place(x=0, y=0)

        # Entry fields and labels for search
        self.order_no_lab = Label(self.root, text="Order number", bg="White")
        self.order_no_lab.grid(row=0, column=0, sticky=E, padx=10, pady=(50,
0))

        self.order_no = Entry(self.root, bd=2, relief=RIDGE)
        self.order_no.bind("<Return>", self.search)
        self.order_no.bind('<FocusIn>', self.destroycal)
        self.order_no.bind('<FocusOut>', lambda
e:self.Num_validate(self.order_no.get(),self.order_no))
        self.order_no.grid(row=0, column=1, sticky=W, padx=(0, 10), pady=(50,
0))

        self.order_date_lab = Label(self.root, text="Order date ",
bg="White")
        self.order_date_lab.grid(row=0, column=2, sticky=E, pady=(50, 0))
        self.date_var = StringVar()
        self.date_var.set("yyyy-mm-dd")

        self.order_date = Entry(self.root, bd=2, relief=RIDGE,
textvariable=self.date_var)
        #self.order_date.bind('<FocusIn>', self.clear_date)
        self.order_date.bind('<Button-1>', lambda e: self.cal_func())
        self.order_date.bind("<Return>", self.search)
        self.order_date.bind('<FocusOut>', lambda e:
self.Date_validate(self.date_var.get(), self.order_date))
        self.order_date.grid(row=0, column=3, sticky=W, pady=(50, 0))

        self.cus_code_lab = Label(self.root, text="Customer code",
bg="White")
        self.cus_code_lab.grid(row=0, column=4, sticky=E, pady=(50, 0))

        self.cus_code = Entry(self.root, bd=2, relief=RIDGE)
        self.cus_code.bind("<Return>", self.search)
        self.cus_code.bind('<FocusIn>', self.destroycal)
        self.cus_code.grid(row=0, column=5, sticky=W, pady=(50, 0))

        # search button

        self.search_btn = Button(self.root, text="Search", font=("times new
roman", 10, "bold"), bg="#1C1B1B", fg="red",
                                 width=10, command=self.search,
relief=RAISED, bd=3)
        self.search_btn.grid(row=1, column=0, columnspan=6, pady=(10, 0))

        # clear button
        self.clear_btn = Button(self.root, text="Clear", font=("times new
roman", 10, "bold"), bg="#1C1B1B", fg="red",
                            width=10, command=self.clear, relief=RAISED,
bd=3)
        self.clear_btn.grid(row=2, column=0, columnspan=6, pady=(10, 0))

        # creating treeview for table
        self.table = ttk.Treeview(self.root, height=10)

        # creating columns
        self.table["columns"] = ("column 2", "column 3", "column 4", "column
5", "column 6", "column 7")

        # formating columns
        self.table.column("#0", width=80, minwidth=65, stretch=NO)
        self.table.column("column 2", width=100, minwidth=90, stretch=NO)
        self.table.column("column 3", width=140, minwidth=120, stretch=NO)
        self.table.column("column 4", width=90, minwidth=70, stretch=NO)
        self.table.column("column 5", width=100, minwidth=80, stretch=NO)
        self.table.column("column 6", width=100, minwidth=80, stretch=NO)
        self.table.column("column 7", width=140, minwidth=120, stretch=NO)

        # defining headings
        self.table.heading("#0", text="ORD_NUM")
        self.table.heading("column 2", text="ORD_AMOUNT")
        self.table.heading("column 3", text="ADVANCE_AMOUNT")
        self.table.heading("column 4", text="ORD_DATE")
        self.table.heading("column 5", text="CUST_CODE")
        self.table.heading("column 6", text="AGENT_CODE")
        self.table.heading("column 7", text="ORD_DESCRIPTION")

        # getting records to insert into treeview
        db.db.cursor.execute(self.query)
        records = db.db.cursor.fetchall()

        # inserting records into treeview
        for i in records:
            self.table.insert("", 'end', text=i[0], values=i[1:])

        # placing the treeview
        self.table.grid(row=3, column=0, columnspan=6, padx=(10, 10),
pady=(20, 0))

        self.root.mainloop()

    def search(self, Event=None):
        self.order_num = self.order_no.get()
        self.ord_dated = self.order_date.get()
        self.cus_cod = self.cus_code.get()

        # 1 search field is used

        if self.order_num != "" and self.ord_dated == "" and self.cus_cod ==
"":
            self.one("ORD_NUM", self.order_num)

        if self.order_num == "" and self.ord_dated != "" and self.cus_cod ==
"":
            self.one("ORD_DATE", self.ord_dated)

        if self.order_num == "" and self.ord_dated == "" and self.cus_cod !=
"":
            self.one("CUST_CODE", self.cus_cod)

        # 2 search fields are used

        if self.order_num != "" and self.ord_dated != "" and self.cus_cod ==
```



```
"":
            self.two("ORD_NUM", "ORD_DATE", self.order_num, self.ord_dated)

        if self.order_num != "" and self.ord_dated == "" and self.cus_cod !=
"":
            self.two("ORD_NUM", "CUST_CODE", self.order_num, self.cus_cod)

        if self.order_num == "" and self.ord_dated != "" and self.cus_cod !=
"":
            self.two("ORD_DATE", "CUST_CODE", self.ord_dated, self.cus_cod)

        # all 3 search fields are used
        if self.order_num != "" and self.ord_dated != "" and self.cus_cod !=
"":
            self.three("ORD_NUM", "ORD_DATE", "CUST_CODE", self.order_num,
self.ord_dated, self.cus_cod)

    def one(self, column, value):

        self.query = "SELECT * FROM `orders` WHERE `%s` = '%s'"

        db.db.cursor.execute(self.query % (column, value))
        records = db.db.cursor.fetchall()

        if records:
            self.table.delete(*self.table.get_children())
            for i in records:
                self.table.insert("", 'end', text=i[0], values=i[1:])
        else:
            messagebox.showerror("error", "No Record Exists")

    def two(self, column1, column2, value1, value2):

        self.query = "SELECT * FROM `orders` WHERE `%s` = '%s' AND `%s` =
'%s'"

        db.db.cursor.execute(self.query % (column1, value1, column2, value2))
        records = db.db.cursor.fetchall()

        if records:
            self.table.delete(*self.table.get_children())
            for i in records:
                self.table.insert("", 'end', text=i[0], values=i[1:])
        else:
            messagebox.showerror("error", "No Record Exists")

    def three(self, column1, column2, column3, value1, value2, value3):

        self.query = "SELECT * FROM `orders` WHERE `%s` = '%s' AND `%s` =
'%s' AND `%s` = '%s'"

        db.db.cursor.execute(self.query % (column1, value1, column2, value2,
column3, value3))
        records = db.db.cursor.fetchall()

        if records:
            self.table.delete(*self.table.get_children())
            for i in records:
                self.table.insert("", 'end', text=i[0], values=i[1:])
        else:
            messagebox.showerror("error", "No Record Exists")

    def clear(self):
        self.order_no.delete(0, 'end')
        self.order_date.delete(0, 'end')
        self.order_date.insert(0,"yyyy-mm-dd")
        self.cus_code.delete(0, 'end')

        # reseting treeview
        self.table.delete(*self.table.get_children())

        # setting query
        self.query = "SELECT * from `orders` WHERE 1"

        # getting records to insert into treeview
        db.db.cursor.execute(self.query)
        records = db.db.cursor.fetchall()

        # inserting records into treeview
        for i in records:
            self.table.insert("", 'end', text=i[0], values=i[1:])
        self.order_date.config(bg="White")
        self.firstclick=1

    def menu(self):
        self.root.destroy()
        x = menu.Menu()
        x.menu()


if __name__ == "__main__":
    x = orders_lookup()
    x.orders()
```

Balance amount module:

```
from tkinter import *
from PIL import ImageTk, Image
from tkinter import messagebox
from tkinter import ttk
import db.db
import menu


class ballance_amt:
    def __init__(self):

        # creating tkinter window
        self.root = Tk()

        self.root.config(bg="white")

        # Setting title
        self.root.title("Sunville Properties | Balance Amt")

        # determining size of window
        self.windowWidth = self.root.winfo_screenwidth() / 2
        self.windowHeight = self.root.winfo_screenheight() / 2

        # detriming the postion to set the window
        self.positionRight = int(self.root.winfo_screenwidth() / 2 -
self.windowWidth / 2)
        self.positionDown = int(self.root.winfo_screenheight() / 2 -
self.windowHeight / 2)

        # Positions the window in the center of the page.
        self.root.geometry("{0}x{1}+{2}+{3}".format(int(self.windowWidth),
int(self.windowHeight), self.positionRight,
                                                    self.positionDown))

        # disable resize of window
        self.root.resizable(False, False)

    def validate_num(self, number):
        try:
            float('%s' % number)
            # messagebox.showinfo("ok", "number")
            return True
        except ValueError:
            return False

    def validate_str(self, string):
        if not (bool(re.search('\d', string))) or string == "":
            regex = re.compile('[@_!#$%^&*()<>?/\|}{~:]')
            if (regex.search(string) == None):
                return True
            else:
                return False
        else:
            return False

    def Num_validate(self, num, widget, event=None):
        if self.validate_num(num) or num == "":
            widget.config(bg="White")
        else:
            widget.config(bg="Red")

    def Str_validate(self, num, widget, event=None):
        if self.validate_str(num):
            widget.config(bg="White")
        else:
            widget.config(bg="Red")

    def balance(self):

        # getting image
        self.image1 = Image.open("D:/python/classes/Internship/property-
consultants-mumbai.jpg")
        self.image1 = self.image1.resize((int(self.windowWidth),
int(self.windowHeight)), Image.ANTIALIAS)
        self.image_bg = ImageTk.PhotoImage(self.image1, master=self.root)

        # placing image
        self.background_label = Label(self.root, image=self.image_bg)
        self.background_label.place(x=0, y=0)
        self.Menu = Button(self.root, text="Menu", bg="Black", fg="Red",
command=self.menu, width=5)
        self.Menu.place(x=0, y=0)

        self.frame1 = Frame(self.root, bg="White")
        self.frame1.pack(pady=10, padx=75, fill=BOTH, expand=True)
        self.frame2 = Frame(self.root, bg="white")
        self.frame2.pack(padx=(0, 320), fill=BOTH, expand=True)

        self.background_label = Label(self.frame2, image=self.image_bg)
        self.background_label.place(x=0, y=-258)

        # creating search fields
        # order num search field
        # order num Label
        self.order_num_lab = Label(self.frame1, text="Order Num", bg="White")
        self.order_num_lab.grid(row=0, column=0, pady=(0, 10))

        self.var_order_num = StringVar()
        self.var_order_num.set("")

        self.order_num = Entry(self.frame1, bd=2, relief=RIDGE, textvariable
=self.var_order_num)
        self.order_num.bind('<Return>', self.search)
        self.order_num.grid(row=0, column=1, pady=(0, 10))

        # agent code
        self.age_code_lab = Label(self.frame1, text="Agent Code", bg="White")
        self.age_code_lab.grid(row=0, column=2, pady=(0, 10))

        self.var_age_code = StringVar()
        self.var_age_code.set("")

        self.age_code = Entry(self.frame1, bd=2, relief=RIDGE, textvariable =
self.var_age_code)
        self.age_code.bind('<Return>', self.search)
        self.age_code.grid(row=0, column=3, pady=(0, 10))

        # Agent name
        self.age_name_lab = Label(self.frame1, text="Agent Name", bg="White")
        self.age_name_lab.grid(row=0, column=4, pady=(0, 10))

        self.var_age_name = StringVar()
        self.var_age_name.set("")

        self.age_name = Entry(self.frame1, bd=2, relief=RIDGE,
textvariable=self.var_age_name)
        self.age_name.bind('<Return>', self.search)
        self.age_name.bind('<FocusOut>', lambda
e:self.Str_validate(self.var_age_name.get(),self.age_name))
        self.age_name.grid(row=0, column=5, pady=(0, 10))

        # search button
        self.search_btn = Button(self.frame1, text = "Search", font =
("calibri",10,"bold"), command=self.search,
                             relief=RAISED, bd=3, bg="Black", fg = "Red",
width = 10)
        self.search_btn.grid(row=1, column=0, columnspan=6)

        # clear button
        self.clear_btn = Button(self.frame1, text = "Clear", font =
("calibri",10,"bold"), command=self.clear,
                             relief=RAISED, bd=3, bg="Black", fg = "Red",
width = 10)
        self.clear_btn.grid(row=2, column=0, columnspan=6, pady=(10, 0))

        # creating treeview for table
        self.table = ttk.Treeview(self.frame1, show="headings", height=5)

        # creating columns
        self.table["columns"] = ("column 1", "column 2", "column 3", "column
4", "column 5", "column 6")

        # formating columns
        self.table.column("column 1", width=80, minwidth=65, stretch=NO)
        self.table.column("column 2", width=100, minwidth=90, stretch=NO)
        self.table.column("column 3", width=140, minwidth=120, stretch=NO)
        self.table.column("column 4", width=90, minwidth=70, stretch=NO)
        self.table.column("column 5", width=100, minwidth=80, stretch=NO)
        self.table.column("column 6", width=100, minwidth=80, stretch=NO)
```

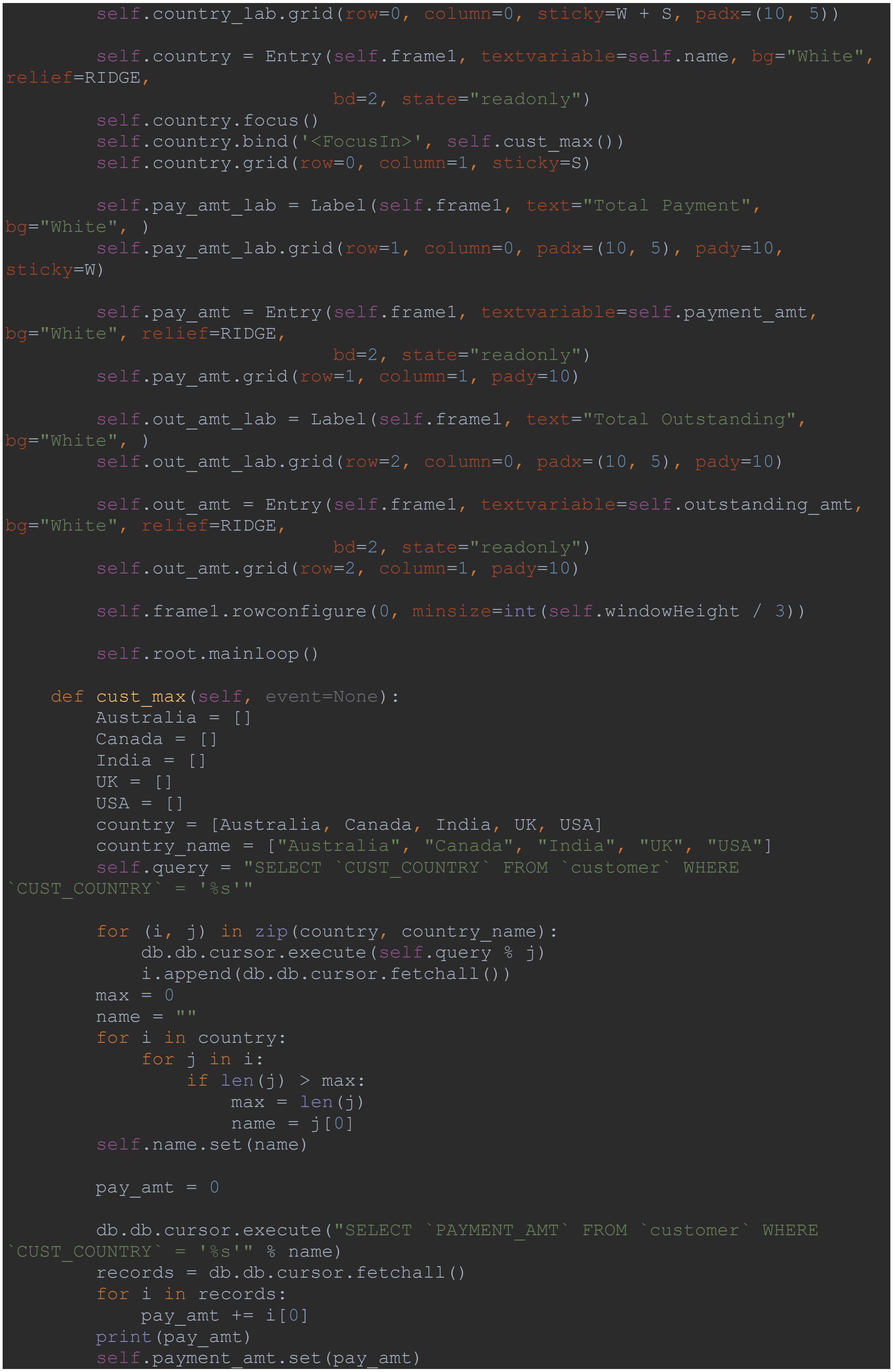


```
        # defining headings
        self.table.heading("column 1", text="ORD_NUM")
        self.table.heading("column 2", text="ORD_AMOUNT")
        self.table.heading("column 3", text="ADVANCE_AMOUNT")
        self.table.heading("column 4", text="BAL_AMOUNT")
        self.table.heading("column 5", text="AGENT_CODE")
        self.table.heading("column 6", text="AGENT_NAME")

        self.query = "SELECT `ORD_NUM` , `ORD_AMOUNT`, `ADVANCE_AMOUNT`,
orders.AGENT_CODE, `AGENT_NAME` FROM orders " \
                     "INNER JOIN agents ON
orders.AGENT_CODE=agents.AGENT_CODE "
        db.db.cursor.execute(self.query)
        records = db.db.cursor.fetchall()
        data = []

        for i in records:
            data.append([i[0], i[1], i[2], i[1] - i[2], i[3], i[4]])

        # inserting records into treeview
        for i in data:
            self.table.insert("", 'end', values=i)

        self.table.bind('<Double 1>', self.get_row)
        self.table.grid(row=3, column=0, columnspan=6)

        # data fields

        self.ord_num_ord = StringVar()
        self.ord_amt_ord = StringVar()
        self.adv_amt_ord = StringVar()
        self.bal_amt_ord = StringVar()
        self.agent_code_ord = StringVar()
        self.agent_name_ord = StringVar()

        # order num

        self.ord_num_lab = Label(self.frame2, text="ORD_NUM", bg="White")
        self.ord_num_lab.grid(row=0, column=0, pady=10)

        self.ord_num = Entry(self.frame2, textvariable=self.ord_num_ord,
bg="White", relief=RIDGE, bd=2)
        self.ord_num.grid(row=0, column=1, pady=10)
        self.ord_num.bind('<FocusOut>', lambda e:
self.Num_validate(self.ord_num_ord.get(), self.ord_num))

        # order amt
        self.ord_amt_lab = Label(self.frame2, text="ORD_AMT", bg="White")
        self.ord_amt_lab.grid(row=1, column=0, pady=10)

        self.ord_amt = Entry(self.frame2, textvariable=self.ord_amt_ord,
bg="White", relief=RIDGE, bd=2)
        self.ord_amt.grid(row=1, column=1, pady=10)
        self.ord_amt.bind('<FocusOut>', lambda e:
self.Num_validate(self.ord_amt_ord.get(), self.ord_amt))

        # adv amt
        self.adv_amt_lab = Label(self.frame2, text="ADVANCE_AMT", bg="White")
        self.adv_amt_lab.grid(row=2, column=0, pady=10)

        self.adv_amt = Entry(self.frame2, textvariable=self.adv_amt_ord,
bg="White", relief=RIDGE, bd=2)
        self.adv_amt.grid(row=2, column=1, pady=10)
        self.adv_amt.bind('<FocusOut>', lambda e:
self.Num_validate(self.adv_amt_ord.get(), self.adv_amt))

        # bal amt
        self.bal_amt_lab = Label(self.frame2, text="BAL_AMT", bg="White")
        self.bal_amt_lab.grid(row=0, column=2, pady=10, padx=(20, 0))

        self.bal_amt = Entry(self.frame2, textvariable=self.bal_amt_ord,
bg="White", relief=RIDGE, bd=2)
        self.bal_amt.grid(row=0, column=3, pady=10)
        self.bal_amt.bind('<FocusOut>', lambda e:
self.Num_validate(self.bal_amt_ord.get(), self.bal_amt))

        # agent code
        self.agent_code_lab = Label(self.frame2, text="AGENT_CODE",
bg="White")
        self.agent_code_lab.grid(row=1, column=2, pady=10, padx=(20, 0))

        self.agent_code = Entry(self.frame2,
textvariable=self.agent_code_ord, bg="White", relief=RIDGE, bd=2)
        self.agent_code.grid(row=1, column=3, pady=10)

        # agent name
        self.agent_name_lab = Label(self.frame2, text="AGENT_NAME",
bg="White")
        self.agent_name_lab.grid(row=2, column=2, pady=10)

        self.agent_name = Entry(self.frame2,
textvariable=self.agent_name_ord, bg="White", relief=RIDGE, bd=2)
        self.agent_name.grid(row=2, column=3, pady=10)
        self.agent_name.bind('<FocusOut>', lambda e:
self.Str_validate(self.agent_name_ord.get(), self.agent_name))

        self.update = Button(self.frame2, text="Update", font=("times new
roman", 10, 'bold'), bg="black",
                             fg="red", command=self.btn_update, width=10)
        self.update.grid(row=3, column=0, columnspan=2, pady=10)

        self.clear_btn1 = Button(self.frame2, text="Clear", font=("times new
roman", 10, 'bold'), bg="black",
                             fg="red", command=self.clear1, width=10)
        self.clear_btn1.grid(row=3, column=2, columnspan=2, pady=10)

        self.root.mainloop()

    def clear1(self):
        self.ord_num_ord.set("")
        self.ord_amt_ord.set("")
        self.adv_amt_ord.set("")
        self.bal_amt_ord.set("")
        self.agent_code_ord.set("")
        self.agent_name_ord.set("")

    def search(self, Event=None):
        self.order_num_var = self.var_order_num.get()
        self.age_code_var = self.var_age_code.get()
        self.age_name_var = self.age_name.get()
        print(self.order_num_var, self.age_code_var, self.age_name_var)
        # 1 search field is used

        if self.order_num_var != "" and self.age_code_var == "" and
self.age_name_var == "":
            self.one("ORD_NUM", self.order_num_var)
        if self.order_num_var == "" and self.age_code_var != "" and
self.age_name_var == "":
            self.one("orders.AGENT_CODE", self.age_code_var)

        if self.order_num_var == "" and self.age_code_var == "" and
self.age_name_var != "":
            self.one("agents.AGENT_NAME", self.age_name_var)

        # 2 search fields are used

        if self.order_num_var != "" and self.age_code_var != "" and
self.age_name_var == "":
            self.two("ORD_NUM", "orders.AGENT_CODE", self.order_num_var,
self.age_code_var)

        if self.order_num_var != "" and self.age_code_var == "" and
self.age_name_var != "":
            self.two("ORD_NUM", "AGENT_NAME", self.order_num,
self.age_name_var)

        if self.order_num_var == "" and self.age_code_var != "" and
self.age_name_var != "":
            self.two("orders.AGENT_CODE", "AGENT_NAME", self.age_code_var,
self.age_name_var)

        # all 3 search fields are used
        if self.order_num_var != "" and self.age_code_var != "" and
self.age_name_var != "":
            self.three("ORD_NUM", "orders.AGENT_CODE", "AGENT_NAME",
self.order_num, self.age_code_var, self.age_name_var)

    def one(self, column, value):
        self.query = "SELECT `ORD_NUM` , `ORD_AMOUNT`, `ADVANCE_AMOUNT`,
orders.AGENT_CODE, `AGENT_NAME` FROM orders " \
                     "INNER JOIN agents ON
orders.AGENT_CODE=agents.AGENT_CODE WHERE %s = '%s' "
        db.db.cursor.execute(self.query % (column, value))
        records = db.db.cursor.fetchall()

        if bool(records):
            self.table.delete(*self.table.get_children())
            data = []

            for i in records:
                data.append([i[0], i[1], i[2], i[1] - i[2], i[3], i[4]])

            # inserting records into treeview
            for i in data:
                self.table.insert("", 'end', values=i)
        elif len(records) ==0:
            messagebox.showerror("error", "No Record Exists")

    def two(self, column1, column2, value1, value2):

        self.query = "SELECT `ORD_NUM` , `ORD_AMOUNT`, `ADVANCE_AMOUNT`,
orders.AGENT_CODE, `AGENT_NAME` FROM orders " \
                     "INNER JOIN agents ON
orders.AGENT_CODE=agents.AGENT_CODE WHERE %s = '%s' AND %s = '%s'"
        db.db.cursor.execute(self.query % (column1, value1, column2, value2))
        records = db.db.cursor.fetchall()

        if records:
            self.table.delete(*self.table.get_children())
            data = []
            for i in records:
                data.append([i[0], i[1], i[2], i[1] - i[2], i[3], i[4]])

            # inserting records into treeview
            for i in data:
                self.table.insert("", 'end', values=i)
        else:
            messagebox.showerror("error", "No Record Exists")

    def three(self, column1, column2, column3, value1, value2, value3):

        self.query = "SELECT `ORD_NUM` , `ORD_AMOUNT`, `ADVANCE_AMOUNT`,
orders.AGENT_CODE, `AGENT_NAME` FROM orders " \
                     "INNER JOIN agents ON
orders.AGENT_CODE=agents.AGENT_CODE WHERE %s = '%s' AND %s = '%s' AND %s =
'%s' "
        db.db.cursor.execute(self.query % (column1, value1, column2, value2,
column3, value3))
        records = db.db.cursor.fetchall()
        if records:
            self.table.delete(*self.table.get_children())
            data = []

            for i in records:
                data.append([i[0], i[1], i[2], i[1] - i[2], i[3], i[4]])

            # inserting records into treeview
            for i in data:
                self.table.insert("", 'end', values=i)
        else:
            messagebox.showerror("error", "No Record Exists")


    def clear(self):
        self.order_num.delete(0, 'end')
        self.age_name.delete(0, 'end')
        self.age_code.delete(0, 'end')

        # reseting treeview
        self.table.delete(*self.table.get_children())

        self.query = "SELECT `ORD_NUM` , `ORD_AMOUNT`, `ADVANCE_AMOUNT`,
orders.AGENT_CODE, `AGENT_NAME` FROM orders " \
                     "INNER JOIN agents ON
orders.AGENT_CODE=agents.AGENT_CODE "
        db.db.cursor.execute(self.query)
        records = db.db.cursor.fetchall()
        data = []

        for i in records:
            data.append([i[0], i[1], i[2], i[1] - i[2], i[3], i[4]])

        # inserting records into treeview
        for i in data:
            self.table.insert("", 'end', values=i)

    def btn_update(self):
        ord_num = self.ord_num_ord.get()
        ord_amt = self.ord_amt_ord.get()
        adv_amt = self.adv_amt_ord.get()

        if self.validate_num(ord_num) and self.validate_num(ord_amt) and
self.validate_num(adv_amt):
```

```
            db.db.cursor.execute("SELECT * FROM `orders` WHERE `ORD_NUM` =
'%s'" % ord_num)
            record = db.db.cursor.fetchall()
            if record:
                response = messagebox.askyesno("Confirmation", "Are you sure
you want to update")
                if response:
                    db.db.cursor.execute(
                        "UPDATE `orders` SET `ORD_AMOUNT` = '%s',
`ADVANCE_AMOUNT` = '%s' WHERE `ORD_NUM` "
                        "= '%s'" % (ord_amt, adv_amt, ord_num))
                    db.db.con.commit()
                    messagebox.showinfo("Success", "Record updated
successfully")
                    # reseting treeview
                    self.table.delete(*self.table.get_children())

                    self.query = "SELECT `ORD_NUM` , `ORD_AMOUNT`,
`ADVANCE_AMOUNT`, orders.AGENT_CODE, `AGENT_NAME` FROM orders " \
                                 "INNER JOIN agents ON
orders.AGENT_CODE=agents.AGENT_CODE "
                    db.db.cursor.execute(self.query)
                    records = db.db.cursor.fetchall()
                    data = []

                    for i in records:
                        data.append([i[0], i[1], i[2], i[1] - i[2], i[3],
i[4]])

                    # inserting records into treeview
                    for i in data:
                        self.table.insert("", 'end', values=i)
            else:
                messagebox.showerror("error", "No such Oreder Number exist")
        else:
            messagebox.showerror("Error", "Error in data entry")

    def get_row(self, event):
        rowid = self.table.identify_row(event.y)
        item = self.table.item(self.table.focus())
        self.ord_num_ord.set(item['values'][0])
        self.ord_amt_ord.set(item['values'][1])
        self.adv_amt_ord.set(item['values'][2])
        self.bal_amt_ord.set(item['values'][3])
        self.agent_code_ord.set(item['values'][4])
        self.agent_name_ord.set(item['values'][5])

    def menu(self):
        self.root.destroy()
        x = menu.Menu()
        x.menu()


if __name__ == "__main__":
    x = ballance_amt()
    x.balance()
```

Customers Module:

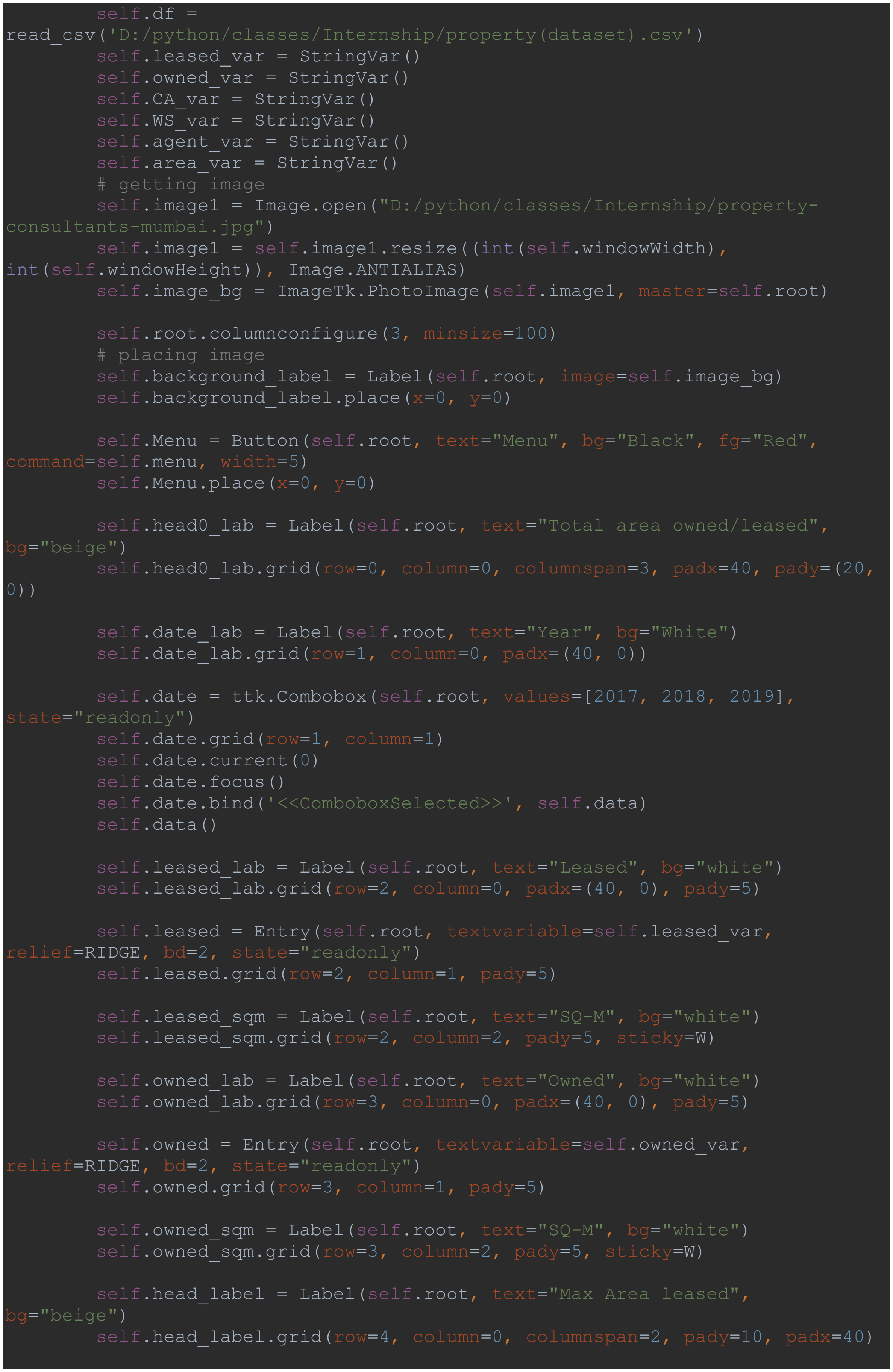
```
from tkinter import *
from PIL import ImageTk, Image
import db.db
import menu


class Customers:
    def __init__(self):
        # creating tkinter window
        self.root = Tk()

        # setting window title
        self.root.title("Sunville Properties | Customers")

        # determining size of window
        self.windowWidth = self.root.winfo_screenwidth() / 2
        self.windowHeight = self.root.winfo_screenheight() / 2

        # determining the the positon to set the window
        self.positionRight = int(self.root.winfo_screenwidth() / 2 -
self.windowWidth / 2)
        self.positionDown = int(self.root.winfo_screenheight() / 2 -
self.windowHeight / 2)

        # Positions the window in the center of the page.
        self.root.geometry(
            "{}x{}+{}+{}".format(int(self.windowWidth),
int(self.windowHeight), self.positionRight, self.positionDown))

        # disable resize of window
        self.root.resizable(False, False)

    def customers(self):
        self.name = StringVar()
        self.payment_amt = StringVar()
        self.outstanding_amt = StringVar()

        # creating frame for customers
        self.frame1 = Frame(self.root, bg="White")
        self.frame1.place(x=0, y=0, width=int(self.windowWidth / 3),
height=int(self.windowHeight))

        # creating frame for image
        self.frame2 = Frame(self.root, bg="blue", )
        self.frame2.place(x=int(self.windowWidth / 3), y=0)

        # getting image
        self.image1 = Image.open("D:/python/classes/Internship/property-
consultants-mumbai.jpg")
        self.image1 = self.image1.resize((int(self.windowWidth / 3 * 2),
int(self.windowHeight)), Image.ANTIALIAS)
        self.image_bg = ImageTk.PhotoImage(self.image1, master=self.root)

        # placing image
        self.background_label = Label(self.frame2, image=self.image_bg)
        self.background_label.pack()

        self.Menu = Button(self.root, text="Menu", bg="Black", fg="Red",
command=self.menu, width=5)
        self.Menu.place(x=0, y=0)

        self.country_lab = Label(self.frame1, text="Max Customers",
bg="White")
```



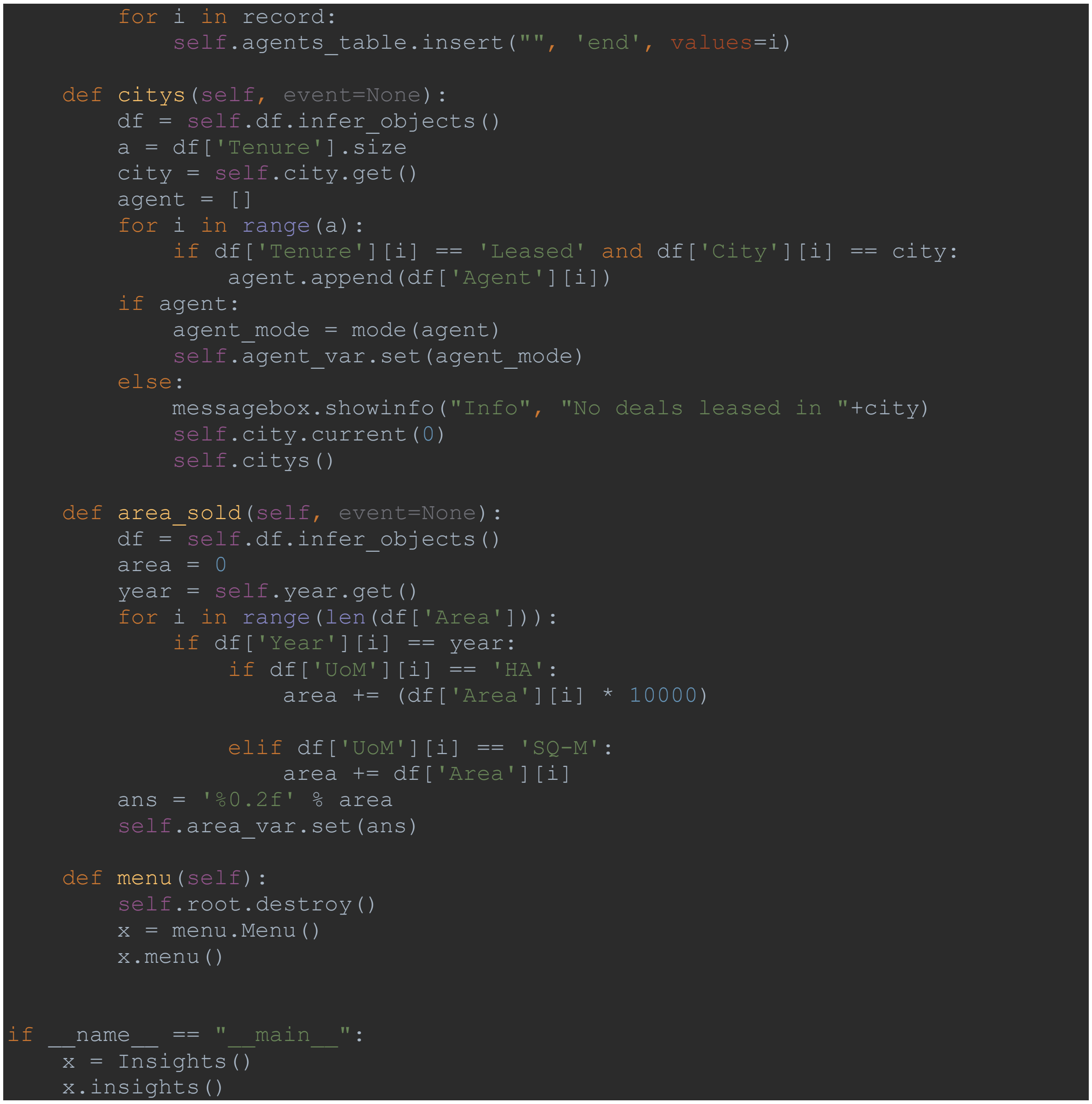
```
        self.country_lab.grid(row=0, column=0, sticky=W + S, padx=(10, 5))

        self.country = Entry(self.frame1, textvariable=self.name, bg="White",
relief=RIDGE,
                             bd=2, state="readonly")
        self.country.focus()
        self.country.bind('<FocusIn>', self.cust_max())
        self.country.grid(row=0, column=1, sticky=S)

        self.pay_amt_lab = Label(self.frame1, text="Total Payment",
bg="White", )
        self.pay_amt_lab.grid(row=1, column=0, padx=(10, 5), pady=10,
sticky=W)

        self.pay_amt = Entry(self.frame1, textvariable=self.payment_amt,
bg="White", relief=RIDGE,
                             bd=2, state="readonly")
        self.pay_amt.grid(row=1, column=1, pady=10)

        self.out_amt_lab = Label(self.frame1, text="Total Outstanding",
bg="White", )
        self.out_amt_lab.grid(row=2, column=0, padx=(10, 5), pady=10)

        self.out_amt = Entry(self.frame1, textvariable=self.outstanding_amt,
bg="White", relief=RIDGE,
                             bd=2, state="readonly")
        self.out_amt.grid(row=2, column=1, pady=10)

        self.frame1.rowconfigure(0, minsize=int(self.windowHeight / 3))

        self.root.mainloop()

    def cust_max(self, event=None):
        Australia = []
        Canada = []
        India = []
        UK = []
        USA = []
        country = [Australia, Canada, India, UK, USA]
        country_name = ["Australia", "Canada", "India", "UK", "USA"]
        self.query = "SELECT `CUST_COUNTRY` FROM `customer` WHERE
`CUST_COUNTRY` = '%s'"

        for (i, j) in zip(country, country_name):
            db.db.cursor.execute(self.query % j)
            i.append(db.db.cursor.fetchall())
        max = 0
        name = ""
        for i in country:
            for j in i:
                if len(j) > max:
                    max = len(j)
                    name = j[0]
        self.name.set(name)

        pay_amt = 0

        db.db.cursor.execute("SELECT `PAYMENT_AMT` FROM `customer` WHERE
`CUST_COUNTRY` = '%s'" % name)
        records = db.db.cursor.fetchall()
        for i in records:
            pay_amt += i[0]
        print(pay_amt)
        self.payment_amt.set(pay_amt)
        out_amt = 0
        db.db.cursor.execute("SELECT `OUTSTANDING_AMT` FROM `customer` WHERE
`CUST_COUNTRY` = '%s'" % name)
        records = db.db.cursor.fetchall()
        for i in records:
            out_amt += i[0]

        self.outstanding_amt.set(out_amt)

    def menu(self):
        self.root.destroy()
        x = menu.Menu()
        x.menu()


if __name__ == "__main__":
    x = Customers()
    x.customers()
```

Insights Module:

```
from tkinter import *
from PIL import ImageTk, Image
from tkinter import ttk
from tkinter import messagebox
from numpy import *
from pandas import *
from statistics import mode
import matplotlib.pyplot as plt
import Graphs
import db.db
import menu


class Insights:

    def __init__(self):
        # creating tkinter window
        self.root = Tk()

        # Setting title
        self.root.title("Sunville Properties | Insights")

        # determining size of the window
        self.windowWidth = self.root.winfo_screenwidth() / 2
        self.windowHeight = self.root.winfo_screenheight() / 2

        # determining the the positon to set the window
        self.positionRight = int(self.root.winfo_screenwidth() / 2 -
self.windowWidth / 2)
        self.positionDown = int(self.root.winfo_screenheight() / 2 -
self.windowHeight / 2)

        # Positions the window in the center of the page.
        self.root.geometry(
            "{}x{}+{}+{}".format(int(self.windowWidth),
int(self.windowHeight), self.positionRight, self.positionDown))

        # disable resize of window
        # self.root.resizable(False, False)

    def insights(self):
```



```
        self.df =
read_csv('D:/python/classes/Internship/property(dataset).csv')
        self.leased_var = StringVar()
        self.owned_var = StringVar()
        self.CA_var = StringVar()
        self.WS_var = StringVar()
        self.agent_var = StringVar()
        self.area_var = StringVar()
        # getting image
        self.image1 = Image.open("D:/python/classes/Internship/property-
consultants-mumbai.jpg")
        self.image1 = self.image1.resize((int(self.windowWidth),
int(self.windowHeight)), Image.ANTIALIAS)
        self.image_bg = ImageTk.PhotoImage(self.image1, master=self.root)

        self.root.columnconfigure(3, minsize=100)
        # placing image
        self.background_label = Label(self.root, image=self.image_bg)
        self.background_label.place(x=0, y=0)

        self.Menu = Button(self.root, text="Menu", bg="Black", fg="Red",
command=self.menu, width=5)
        self.Menu.place(x=0, y=0)

        self.head0_lab = Label(self.root, text="Total area owned/leased",
bg="beige")
        self.head0_lab.grid(row=0, column=0, columnspan=3, padx=40, pady=(20,
0))

        self.date_lab = Label(self.root, text="Year", bg="White")
        self.date_lab.grid(row=1, column=0, padx=(40, 0))

        self.date = ttk.Combobox(self.root, values=[2017, 2018, 2019],
state="readonly")
        self.date.grid(row=1, column=1)
        self.date.current(0)
        self.date.focus()
        self.date.bind('<<ComboboxSelected>>', self.data)
        self.data()

        self.leased_lab = Label(self.root, text="Leased", bg="white")
        self.leased_lab.grid(row=2, column=0, padx=(40, 0), pady=5)

        self.leased = Entry(self.root, textvariable=self.leased_var,
relief=RIDGE, bd=2, state="readonly")
        self.leased.grid(row=2, column=1, pady=5)

        self.leased_sqm = Label(self.root, text="SQ-M", bg="white")
        self.leased_sqm.grid(row=2, column=2, pady=5, sticky=W)

        self.owned_lab = Label(self.root, text="Owned", bg="white")
        self.owned_lab.grid(row=3, column=0, padx=(40, 0), pady=5)

        self.owned = Entry(self.root, textvariable=self.owned_var,
relief=RIDGE, bd=2, state="readonly")
        self.owned.grid(row=3, column=1, pady=5)

        self.owned_sqm = Label(self.root, text="SQ-M", bg="white")
        self.owned_sqm.grid(row=3, column=2, pady=5, sticky=W)

        self.head_label = Label(self.root, text="Max Area leased",
bg="beige")
        self.head_label.grid(row=4, column=0, columnspan=2, pady=10, padx=40)
        self.CA_lab = Label(self.root, text="CA Countries", bg="White")
        self.CA_lab.grid(row=5, column=0, pady=5, padx=(40, 0))

        self.CA = Entry(self.root, textvariable=self.CA_var, relief=RIDGE,
bd=2, state="readonly")
        self.CA.grid(row=5, column=1, pady=5)

        self.CA_sqm = Label(self.root, text="SQ-M", bg="white")
        self.CA_sqm.grid(row=5, column=2, pady=5, sticky=W)

        self.WS_lab = Label(self.root, text="WS Countries", bg="White")
        self.WS_lab.grid(row=6, column=0, pady=5, padx=(40, 0))

        self.WS = Entry(self.root, textvariable=self.WS_var, relief=RIDGE,
bd=2, state="readonly")
        self.WS.grid(row=6, column=1, pady=5)

        self.WS_sqm = Label(self.root, text="SQ-M", bg="white")
        self.WS_sqm.grid(row=6, column=2, pady=5, sticky=W)
        self.CA_WS()

        self.head2_lab = Label(self.root, text="Agents with deals as owned",
bg="beige")
        self.head2_lab.grid(row=7, column=0, columnspan=7, pady=10, padx =
(150,0))

        self.agents_table = ttk.Treeview(self.root, height=5,
show="headings")
        self.agents_table['columns'] = ['column 1', 'column 2', 'column 3',
'column 4']

        self.agents_table.column("column 1", width=80, minwidth=65,
stretch=NO)
        self.agents_table.column("column 2", width=100, minwidth=90,
stretch=NO)
        self.agents_table.column("column 3", width=100, minwidth=90,
stretch=NO)
        self.agents_table.column("column 4", width=90, minwidth=70,
stretch=NO)

        self.agents_table.heading("column 1", text="AGENT_CODE")
        self.agents_table.heading("column 2", text="AGENT_NAME")
        self.agents_table.heading("column 3", text="WORKING_AREA")
        self.agents_table.heading("column 4", text="PHONE_NO")
        self.agents_table.place(x=200, y=300)
        #self.agents_table.grid(row=8, column=1, columnspan=4, rowspan=6,
padx=(50,0))

        self.agents()

        self.head3_lab = Label(self.root, text="Maximums deals leased",
bg="beige")
        self.head3_lab.grid(row=0, column=4, padx=10, pady=(20, 5),
columnspan=3)
        self.city_lab = Label(self.root, text="City", bg="White")
        self.city_lab.grid(row=1, column=4, pady=5)

        df = self.df.infer_objects()

        a = df['City'].size
        my_dict = {}
        city1 = []
        for i in range(a):
            my_dict[df['City'][i]] = i
        for keys in my_dict.keys():
            city1.append(keys)
        c = city1.index('Chilliwack')
        city = []
        city.append(city1[c])
        city1.pop(c)
        for citys in city1:
            city.append(citys)

        self.city = ttk.Combobox(self.root, values=city, state='readonly',
font=('times new roman', 10, ''))
        self.city.grid(row=1, column=5, pady=5)
        self.city.current(0)
        self.city.bind('<<ComboboxSelected>>', self.citys)

        self.agent_lab = Label(self.root, text="Agent", bg="White")
        self.agent_lab.grid(row=2, column=4, pady=5)
        self.agent = Entry(self.root, textvariable=self.agent_var,
relief=RIDGE, bd=2, state="readonly")
        self.agent.grid(row=2, column=5, pady=5)
        self.citys()

        '''self.head4_lab = Label(self.root, text="Agent performance",
bg="beige")
        self.head4_lab.grid(row=3, column=4, columnspan=3, pady=5)

        self.year_agent_lab = Label(self.root, text="Year", bg="White")
        self.year_agent_lab.grid(row=4, column=4)

        agent_year= ['2017', '2018', '2019', '2020', '2017-2020']
        years = [2017, 2018, 2019, 2020]

        self.year_agent = ttk.Combobox(self.root, values= agent_year,
state='readonly')
        self.year_agent.current(0)
        self.year_agent.bind('<Return>', self.agent_performance)
        self.year_agent.grid(row=4, column=5)

        self.agent_perf = Button(self.root, text="Agent Performance",
bg="Black", fg="Red", width=15, relief=RIDGE,
                                 font=("times new roman", 10, 'bold'),
command=self.agent_performance)
        self.agent_perf.grid(row=5, column=4, columnspan=3, pady=5)'''

        self.head5_lab = Label(self.root, text="Area sold in July",
bg="beige")
        self.head5_lab.grid(row=4, column=5, columnspan=3, pady=5)

        years = [2017, 2018, 2019, 2020]

        self.year_lab = Label(self.root, text="Year", bg="White")
        self.year_lab.grid(row=5, column=4)

        self.year = ttk.Combobox(self.root, values=years, state='readonly')
        self.year.current(0)
        self.year.bind('<<ComboboxSelected>>', self.area_sold)
        self.year.grid(row=5, column=5)
        self.area_sold()

        self.area_lab = Label(self.root, text="Area Sold", bg="white")
        self.area_lab.grid(row=6, column=4)

        self.area = Entry(self.root, textvariable=self.area_var, bg="White",
relief=RIDGE, state='readonly', bd=2)
        self.area.grid(row=6, column=5)

        self.area_sqm = Label(self.root, text="SQ-M", bg="white")
        self.area_sqm.grid(row=6, column=7, pady=5, sticky=W)

        '''self.head6_lab = Label(self.root, text="Time analysis of orders",
bg="beige")
        self.head6_lab.grid(row=9, column=4, columnspan=3, pady=5)

        self.time_ana = Button(self.root, text="Time Analysis", bg="Black",
fg="Red", width=15, relief=RIDGE,
                               font=("times new roman", 10, 'bold'),
command=self.time_analysis)
        self.time_ana.grid(row=10, column=4, columnspan=3, pady=5)'''

        self.next_btn = Button(self.root, text="Next", bg="Black", fg="Red",
width=10, relief=RIDGE,
                               font=("times new roman", 10, 'bold'),
command=self.next)
        self.next_btn.place(x = int(self.windowWidth)-80, y
=int(self.windowHeight)-25)

        self.root.mainloop()

    def next(self):
        self.root.destroy()
        gra = Graphs.Graphs()
        gra.graphs()

    def data(self, event=None):

        year = self.date.get()
        leased = 0
        owned = 0
        df = self.df.infer_objects()
        a = df['Tenure'].size
        for i in range(a):
            if df['Year'][i] == year:
                if df['Tenure'][i] == "Leased":
                    if df['UoM'][i] == 'HA':
                        leased += (df['Area'][i] * 10000)
                    else:
                        leased += df['Area'][i]
                else:
                    if df['UoM'][i] == 'HA':
                        owned += (df['Area'][i] * 10000)
                    else:
                        owned += df['Area'][i]
        self.leased_var.set('%.2f' % leased)
        self.owned_var.set('%.2f' % owned)

    def CA_WS(self):

        sum17 = 0
        sum18 = 0
        sum19 = 0
        sum20 = 0
        sum17_WS = 0
        sum18_WS = 0
        sum19_WS = 0
        sum20_WS = 0
        df = self.df.infer_objects()
        a = df['Country'].size
        for i in range(a):
            if df['Country'][i] == "CA":
                if df['Year'][i] == '2017':
                    if df['UoM'][i] == 'HA':
                        sum17 += (df['Area'][i] * 10000)
                    else:
                        sum17 += df['Area'][i]
                elif df['Year'][i] == '2018':
                    if df['UoM'][i] == 'HA':
                        sum18 += (df['Area'][i] * 10000)
                    else:
                        sum18 += df['Area'][i]
                elif df['Year'][i] == '2019':
                    if df['UoM'][i] == 'HA':
                        sum19 += (df['Area'][i] * 10000)
                    else:
                        sum19 += df['Area'][i]
                elif df['Year'][i] == '2020':
                    if df['UoM'][i] == 'HA':
                        sum20 += (df['Area'][i] * 10000)
                    else:
                        sum20 += df['Area'][i]
            elif df['Country'][i] == "WS":
                if df['Year'][i] == '2017':
                    if df['UoM'][i] == 'HA':
                        sum17_WS += (df['Area'][i] * 10000)
                    else:
                        sum17_WS += df['Area'][i]
                elif df['Year'][i] == '2018':
                    if df['UoM'][i] == 'HA':
                        sum18_WS += (df['Area'][i] * 10000)
                    else:
                        sum18_WS += df['Area'][i]
                elif df['Year'][i] == '2019':
                    if df['UoM'][i] == 'HA':
                        sum19_WS += (df['Area'][i] * 10000)
                    else:
                        sum19_WS += df['Area'][i]
                elif df['Year'][i] == '2020':
                    if df['UoM'][i] == 'HA':
                        sum20_WS += (df['Area'][i] * 10000)
                    else:
                        sum20_WS += df['Area'][i]
        ans_c = '%.2f' % max(sum20, sum19, sum18, sum17)
        ans_w = '%.2f' % max(sum20_WS, sum19_WS, sum18_WS, sum17_WS)
        self.CA_var.set(ans_c)
        self.WS_var.set(ans_w)

    def agents(self):
        df = self.df.infer_objects()
        a = df['Tenure'].size
        my_dict = {}
        my_list = []
        for i in range(a):
            if df['Tenure'][i] == 'Owned':
                my_dict[df['Agent'][i]] = df['Identifier'][i]
        for keys in my_dict.keys():
            my_list.append(keys)

        db.db.cursor.execute("SELECT `AGENT_CODE`, `AGENT_NAME`,
`WORKING_AREA`, `PHONE_NO` FROM `agents` WHERE "
                             "`AGENT_NAME` IN " + str(tuple(tuple(my_list))))
        record = db.db.cursor.fetchall()
```



```
        for i in record:
            self.agents_table.insert("", 'end', values=i)

    def citys(self, event=None):
        df = self.df.infer_objects()
        a = df['Tenure'].size
        city = self.city.get()
        agent = []
        for i in range(a):
            if df['Tenure'][i] == 'Leased' and df['City'][i] == city:
                agent.append(df['Agent'][i])
        if agent:
            agent_mode = mode(agent)
            self.agent_var.set(agent_mode)
        else:
            messagebox.showinfo("Info", "No deals leased in "+city)
            self.city.current(0)
            self.citys()

    def area_sold(self, event=None):
        df = self.df.infer_objects()
        area = 0
        year = self.year.get()
        for i in range(len(df['Area'])):
            if df['Year'][i] == year:
                if df['UoM'][i] == 'HA':
                    area += (df['Area'][i] * 10000)

                elif df['UoM'][i] == 'SQ-M':
                    area += df['Area'][i]
        ans = '%0.2f' % area
        self.area_var.set(ans)

    def menu(self):
        self.root.destroy()
        x = menu.Menu()
        x.menu()


if __name__ == "__main__":
    x = Insights()
    x.insights()
```

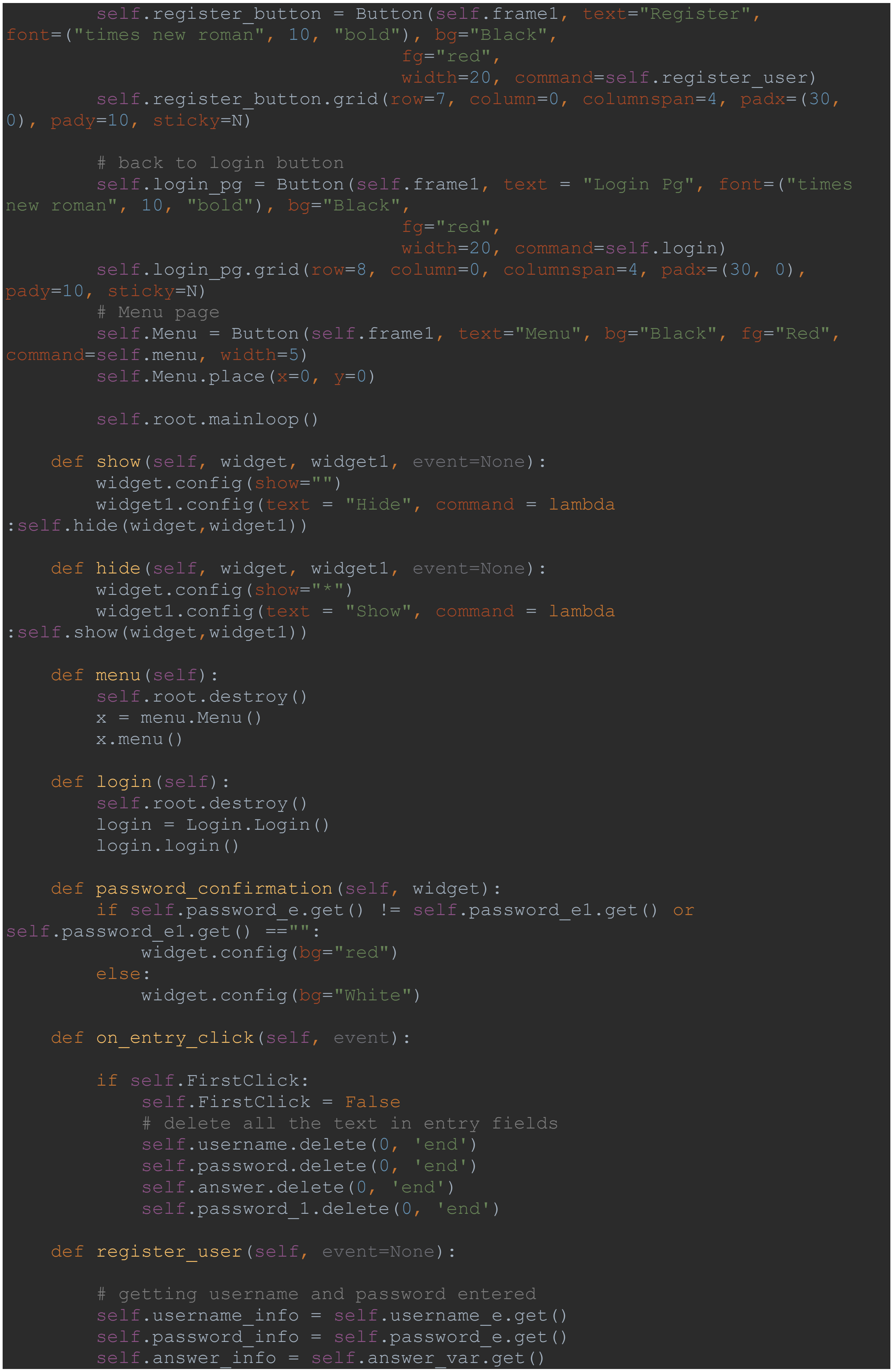
Graphs Module:

```
from tkinter import *
from PIL import Image, ImageTk
from tkinter import ttk
from numpy import *
from pandas import *
from statistics import mode
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.backends.backend_tkagg import NavigationToolbar2Tk
import insights
import db.db
import menu

class Graphs:
    def __init__(self):
        # creating tkinter window
        self.root = Tk()

        # setting window title
        self.root.title("Sunville Properties | Insights | Graphs")

        # determining size of window
        self.windowWidth = self.root.winfo_screenwidth() / 2
        self.windowHeight = self.root.winfo_screenheight() / 2

        # determining the the positon to set the window
        self.positionRight = int(self.root.winfo_screenwidth() / 2 -
self.windowWidth / 2)
        self.positionDown = int(self.root.winfo_screenheight() / 2 -
self.windowHeight / 2)

        # Positions the window in the center of the page.
        self.root.geometry(
            "{}x{}+{}+{}".format(int(self.windowWidth),
int(self.windowHeight), self.positionRight, self.positionDown))

        # disable resize of window
        self.root.resizable(False, False)

    def graphs(self):
        self.df =
read_csv('D:/python/classes/Internship/property(dataset).csv')

        # getting image
        self.image1 = Image.open("D:/python/classes/Internship/property-
consultants-mumbai.jpg")
        self.image1 = self.image1.resize((int(self.windowWidth),
int(self.windowHeight)), Image.ANTIALIAS)
        self.image_bg = ImageTk.PhotoImage(self.image1, master=self.root)

        self.root.columnconfigure(3, minsize=100)
        # placing image
        self.background_label = Label(self.root, image=self.image_bg)
        self.background_label.place(x=0, y=0)

        self.Menu = Button(self.root, text="Menu", bg="Black", fg="Red",
command=self.menu, width=5)
        self.Menu.place(x=0, y=0)

        self.Back = Button(self.root, text="Back", bg="Black", fg="Red",
command=self.back, width=5)
```

```
        self.Back.place(x=0, y=30)

        self.parent_frame = Frame(self.root, bg = "White")
        self.parent_frame.pack()


        self.graph_chg = Button(self.parent_frame, text="Time Series",
bg="Black", fg="Red", command=lambda :self.next(self.graph_chg), width=10)
        self.graph_chg.pack(side = LEFT, anchor=N)

        self.frame = Frame(self.parent_frame, bg = "White")
        self.frame.pack()

        self.frame1 = LabelFrame(self.frame, text = "Agent performance", bg =
"White")
        self.frame1.pack(side = LEFT, expand = True, fill = BOTH)

        agent_year = ['2017', '2018', '2019', '2020', '2017-2020']
        self.year_agent = ttk.Combobox(self.frame1, values=agent_year,
state='readonly')
        self.year_agent.current(0)
        self.year_agent.bind('<<ComboboxSelected>>', self.agent_performance)
        self.year_agent.pack()

        self.agent_performance()

        self.root.mainloop()

    def agent_performance(self, event=None):
        year = self.year_agent.get()
        df = self.df.infer_objects()
        db.db.cursor.execute("SELECT `AGENT_NAME` from `agents`")
        records = db.db.cursor.fetchall()
        self.agents_list = []
        area = 0
        self.area_list = []
        a = df['Tenure'].size
        for i in records:
            for j in i:
                self.agents_list.append(j)

        if year == '2017' or year == '2018' or year == '2019' or year ==
'2020':

            try:
                self.canvas.get_tk_widget().pack_forget()
                self.toolbar.pack_forget()
            except AttributeError:
                pass
            for i in self.agents_list:
                area = 0
                for j in range(a):
                    if df['Year'][j] == year:
                        if df['Agent'][j] == i:
                            if df['UoM'][j] == 'HA':
                                area += (df['Area'][j] * 10000)

                            elif df['UoM'][j] == 'SQ-M':
                                area += df['Area'][j]
                self.area_list.append('%0.2f' % area)
            self.area_list = [float(i) for i in self.area_list]

            x = arange(len(self.agents_list))
            f = plt.Figure(figsize=(5,5), dpi=100, tight_layout=True)
            a = f.add_subplot(111)
            a.bar(x, self.area_list)
            a.set_xticks(x)
            a.set_xticklabels(self.agents_list, rotation=60)
            a.set_xlabel('Agents')
            a.set_ylabel('Area(owned+leased) SQ-M')


            #plt.show()
            self.canvas = FigureCanvasTkAgg(f, self.frame1)
            self.canvas.draw()
            self.toolbar = NavigationToolbar2Tk(self.canvas, self.frame1)
            self.toolbar.update()
            self.canvas.get_tk_widget().pack(side=TOP, fill=BOTH,
expand=True)



        elif year == '2017-2020':
            try:
                self.canvas.get_tk_widget().pack_forget()
                self.toolbar.pack_forget()
            except AttributeError:
                pass
            for i in self.agents_list:
                area = 0
                for j in range(a):
                    if df['Agent'][j] == i:
                        if df['UoM'][j] == 'HA':
                            area += (df['Area'][j] * 10000)

                        elif df['UoM'][j] == 'SQ-M':
                            area += df['Area'][j]
                self.area_list.append('%0.2f' % area)
            self.area_list = [float(i) for i in self.area_list]

            x = arange(len(self.agents_list))
            f = plt.Figure(figsize=(5,6), dpi=100)
            a = f.add_subplot(111)
            f.subplots_adjust(bottom =0.338, top = 0.945, left=0.171,
right=0.94)
            a.bar(x, self.area_list)
            a.set_xticks(x)
            a.set_xticklabels(self.agents_list, rotation=60)
            a.set_xlabel('Agents')
            a.set_ylabel('Area(owned+leased) SQ-M')

            # plt.show()
            self.canvas = FigureCanvasTkAgg(f, self.frame1)
            self.canvas.draw()
            self.toolbar = NavigationToolbar2Tk(self.canvas, self.frame1)
            self.toolbar.update()
            self.canvas.get_tk_widget().pack()


    def time_analysis(self):

        try:
            self.canvas1.get_tk_widget().pack_forget()
            self.toolbar1.pack_forget()
        except AttributeError:
            pass

        db.db.cursor.execute("SELECT `ORD_DATE` FROM `orders` ORDER BY
`orders`.`ORD_DATE` ASC")
        records1 = db.db.cursor.fetchall()

        date = []

        for i in records1:
            for j in i:
                date.append(j)

        db.db.cursor.execute("SELECT `ORD_AMOUNT` FROM `orders` ORDER BY
`orders`.`ORD_DATE` ASC")
        records2 = db.db.cursor.fetchall()

        ord_amt = []

        for i in records2:
            for j in i:
                ord_amt.append(j)
        ord_amt = [float(i) for i in ord_amt]

        #plt.plot_date(date, ord_amt, linestyle = 'solid')
        #plt.gcf().autofmt_xdate()
        #plt.tight_layout()
        fig = plt.Figure(figsize=(5.5,5),dpi=100)
        fig.add_subplot(111).plot(date, ord_amt,"bo", linestyle = "solid")
        fig.subplots_adjust(top=0.93, left=0.09, right=0.988)
        fig.autofmt_xdate()
        self.canvas1 = FigureCanvasTkAgg(fig, self.frame2)
        self.canvas1.draw()
        self.toolbar1 = NavigationToolbar2Tk(self.canvas1, self.frame2)
        self.toolbar1.update()
        self.canvas1.get_tk_widget().pack()
        #plt.show()

    def menu(self):
        self.root.destroy()
        x = menu.Menu()
        x.menu()

    def back(self):
        self.root.destroy()
        x = insights.Insights()
        x.insights()
    def next(self, widget):
        for widgets in self.frame.winfo_children():
            widgets.destroy()
        self.frame2 = LabelFrame(self.frame, text="Time series analysis of
orders", bg = "White")
        self.frame2.pack(side=LEFT, expand=True, fill=BOTH)

        widget.config(text="Agent performance", width=15, command=lambda
:self.next1(widget))
        self.time_analysis()

    def next1(self, widget):
        for widgets in self.frame.winfo_children():
            widgets.destroy()

        self.frame1 = LabelFrame(self.frame, text="Agent performance", bg =
"White")
        self.frame1.pack(side=LEFT, expand=True, fill=BOTH)

        agent_year = ['2017', '2018', '2019', '2020', '2017-2020']
        self.year_agent = ttk.Combobox(self.frame1, values=agent_year,
state='readonly')
        self.year_agent.current(0)
        self.year_agent.bind('<<ComboboxSelected>>', self.agent_performance)
        self.year_agent.pack()

        widget.config(text="Time Series", width=10, command=lambda:
self.next(widget))
        self.agent_performance()


if __name__ == "__main__":
    x = Graphs()
    x.graphs()
```

Register module:

```
from tkinter import *
from PIL import ImageTk, Image
from tkinter import messagebox
from tkinter import ttk
import db.db
import Login
import menu


class Register:
    def __init__(self):

        # creating tkinter window
        self.root = Tk()

        # setting window title
        self.root.title("Sunville Properties | Register")

        # determining size of window
        self.windowWidth = self.root.winfo_screenwidth() / 2
        self.windowHeight = self.root.winfo_screenheight() / 2

        # determining the the positon to set the window
        self.positionRight = int(self.root.winfo_screenwidth() / 2 -
self.windowWidth / 2)
        self.positionDown = int(self.root.winfo_screenheight() / 2 -
self.windowHeight / 2)

        # Positions the window in the center of the page.
        self.root.geometry(
            "{}x{}+{}+{}".format(int(self.windowWidth),
int(self.windowHeight), self.positionRight, self.positionDown))

        # disable resize of window
        self.root.resizable(False, False)

        self.FirstClick = True

    def register(self):

        # variable of entry fields
        self.username_e = StringVar()
        self.username_e.set("Username")
        self.password_e = StringVar()
        self.password_e.set("password")
        self.answer_var = StringVar()
        self.answer_var.set("Answer")
        self.password_e1 = StringVar()
        self.password_e1.set("password")

        # creating frame for register form
        self.frame1 = Frame(self.root, bg="White")
        self.frame1.place(x=0, y=0, width=int(self.windowWidth / 2),
height=int(self.windowHeight))

        # creating frame for image
        self.frame2 = Frame(self.root, bg="blue", )
        self.frame2.place(x=int(self.windowWidth / 2), y=0)

        # getting image
        self.image1 = Image.open("D:/python/classes/Internship/property-
consultants-mumbai.jpg")
        self.image1 = self.image1.resize((int(self.windowWidth / 2),
int(self.windowHeight)), Image.ANTIALIAS)
        self.image_bg = ImageTk.PhotoImage(self.image1, master=self.root)

        # placing image
        self.background_label = Label(self.frame2, image=self.image_bg)
        self.background_label.pack()

        # heading
        self.heading = Label(self.frame1, text="Sunville Properties",
font=("corbel", 15, "bold italic"), bg="beige",
                             fg="red")
        self.heading.grid(row=0, column=0, columnspan=4, ipadx=5)

        # login image for entry form
        self.image2 = Image.open("D:/python/classes/Internship/user.png")
        self.image2 = self.image2.resize((25, 25), Image.ANTIALIAS)
        self.login_img = ImageTk.PhotoImage(self.image2, master = self.root)

        # placing login image and heading
        self.label1 = Label(self.frame1, image=self.login_img,
text="Register", compound=LEFT,
                            font=("calibri", 15, "bold"), pady=10,
anchor=CENTER, bg='White')
        self.label1.grid(row=1, column=0, columnspan=4, padx=(30, 0),
sticky=S)

        self.frame1.rowconfigure(0, minsize=int(self.windowHeight / 4))

        # username label
        self.username_lab = Label(self.frame1, text="Username",
font=("calibri", 10, "bold"), bg="White")
        self.username_lab.grid(row=2, column=0, pady=(0, 10))

        # username entry field
        self.username = Entry(self.frame1, textvariable=self.username_e,
relief=SUNKEN, bd=2, )
        self.username.bind('<FocusIn>', self.on_entry_click)
        self.username.bind('<Return>', self.register_user)
        self.username.grid(row=2, column=1, columnspan=3, pady=(0, 10))

        # Question label
        self.question_lab = Label(self.frame1, text="question",
font=("calibri", 10, "bold"), bg="white")
        self.question_lab.grid(row=3, column=0, pady = (0,10), padx = (10,
20))

        # questions list
        questions = ["What is your mother's maiden name", "What is the name
of your first pet", "What is the name of "
"your first pet",
                     "What is your father's middle name"]

        # questions drop down
        self.question = ttk.Combobox(self.frame1, width = 33, values =
questions, state = "readonly")
        self.question.grid(row=3, column=3, columnspan=5, pady = (0, 10))
        self.question.current(0)

        # answer Label
        self.answer_lab = Label(self.frame1, text="Answer", font=("calibri",
10, "bold"), bg="white")
        self.answer_lab.grid(row = 4, column = 0, padx = (20, 10), pady = (0,
10))

        # answer entry field
        self.answer = Entry(self.frame1, textvariable = self.answer_var,
relief=SUNKEN, bd=2)
        self.answer.bind('<FocusIn>', self.on_entry_click)
        self.answer.bind('<Return>', self.register_user)
        self.answer.grid(row=4, column=1, columnspan=3, pady = (0, 10))

        # password label
        self.password_lab = Label(self.frame1, text="Password",
font=("calibri", 10, "bold"), bg="white")
        self.password_lab.grid(row=5, column=0, padx=(20, 10), pady=(0, 10))

        # password entry field
        self.password = Entry(self.frame1, textvariable=self.password_e,
relief=SUNKEN, bd=2, show="*")
        self.password.bind('<FocusIn>', self.on_entry_click)
        self.password.bind('<Return>', self.register_user)
        self.password.grid(row=5, column=1, columnspan=3)

        # show password button1
        self.show_btn1 = Button(self.frame1, text="Show", font=("times new
roman", 10, "bold"), bg="Black",
                                fg="red", width=7, command=lambda:
self.show(self.password, self.show_btn1))
        self.show_btn1.grid(row=5, column=4)

        # password confirmation label
        self.password_lab1 = Label(self.frame1, text="Password",
font=("calibri", 10, "bold"), bg="white")
        self.password_lab1.grid(row=6, column=0, padx=(20, 10), pady=(0, 10))

        # password confirmation entry field
        self.password_1 = Entry(self.frame1, textvariable=self.password_e1,
relief=SUNKEN, bd=2, show="*")
        self.password_1.bind('<FocusIn>', self.on_entry_click)
        self.password_1.bind('<FocusOut>', lambda e:
self.password_confirmation(self.password_1))
        self.password_1.bind('<Return>', self.register_user)
        self.password_1.grid(row=6, column=1, columnspan=3)

        # show password button2
        self.show_btn2 = Button(self.frame1, text="Show", font=("times new
roman", 10, "bold"), bg="Black",
                                fg="red", width=7, command=lambda:
self.show(self.password_1, self.show_btn2))
        self.show_btn2.grid(row=6, column=4)

        # register button
```



```
        self.register_button = Button(self.frame1, text="Register",
font=("times new roman", 10, "bold"), bg="Black",
                                   fg="red",
                                   width=20, command=self.register_user)
        self.register_button.grid(row=7, column=0, columnspan=4, padx=(30,
0), pady=10, sticky=N)

        # back to login button
        self.login_pg = Button(self.frame1, text = "Login Pg", font=("times
new roman", 10, "bold"), bg="Black",
                                   fg="red",
                                   width=20, command=self.login)
        self.login_pg.grid(row=8, column=0, columnspan=4, padx=(30, 0),
pady=10, sticky=N)
        # Menu page
        self.Menu = Button(self.frame1, text="Menu", bg="Black", fg="Red",
command=self.menu, width=5)
        self.Menu.place(x=0, y=0)

        self.root.mainloop()

    def show(self, widget, widget1, event=None):
        widget.config(show="")
        widget1.config(text = "Hide", command = lambda
:self.hide(widget,widget1))

    def hide(self, widget, widget1, event=None):
        widget.config(show="*")
        widget1.config(text = "Show", command = lambda
:self.show(widget,widget1))

    def menu(self):
        self.root.destroy()
        x = menu.Menu()
        x.menu()

    def login(self):
        self.root.destroy()
        login = Login.Login()
        login.login()

    def password_confirmation(self, widget):
        if self.password_e.get() != self.password_e1.get() or
self.password_e1.get() =="":
            widget.config(bg="red")
        else:
            widget.config(bg="White")

    def on_entry_click(self, event):

        if self.FirstClick:
            self.FirstClick = False
            # delete all the text in entry fields
            self.username.delete(0, 'end')
            self.password.delete(0, 'end')
            self.answer.delete(0, 'end')
            self.password_1.delete(0, 'end')

    def register_user(self, event=None):

        # getting username and password entered
        self.username_info = self.username_e.get()
        self.password_info = self.password_e.get()
        self.answer_info = self.answer_var.get()
        self.password_info1 = self.password_e1.get()
        self.question_info = self.question.get()

        # checking if all fields are full
        if self.username_info == "":
            messagebox.showerror("error", "username can not be blank")
            self.username_e.set("Username")
            self.password_e.set("Password")
            self.password_e1.set("Password")
            self.answer_var.set("Answer")
            self.FirstClick = True
            self.username_lab.focus()

        elif self.answer_info =="":
            messagebox.showerror("error", "Answer can not be blank")
            self.answer.focus()

        elif self.password_info == "":
            messagebox.showerror("error", "Password can not be blank")
            self.password.focus()

        elif self.password_info1 == "":
            messagebox.showerror("error", "Password can not be blank")
            self.password_1.focus()

        else:

            db.db.cursor.execute("SELECT * FROM `login` WHERE `Username` =
'%s'" % self.username_info)
            records = db.db.cursor.fetchall()

            if records:
                messagebox.showerror("Error", "Username already exist")

            else:
                if self.password_info == self.password_info1:
                    query = """INSERT INTO `login`(`Username`, `Password`,
`Question`, `Answer`)
                    VALUES ("%s",md5("%s"), "%s","%s")"""

                    db.db.cursor.execute(query % (self.username_info,
self.password_info, self.question_info, self.answer_info))
                    db.db.con.commit()
                    messagebox.showinfo("Success", "Registered successfully")
                else:
                    messagebox.showerror("Error", "Passwords don't match")



if __name__ == "__main__":
    x = Register()
    x.register()
```

CONCLUSION

The project entitled “Sunville Properties App” is developed using Python Tkinter as front-end MySQL database in the back end to computerize and ease the process of Sunville Properties, which can help them seamlessly navigate via different forms of storage and which can also help them to fetch, modify and analyse their data.